Demeter Marketplace Technical Report

1. Executive Summary

The Demeter Marketplace platform is engineered as a scalable, cloud-native solution on Amazon Web Services (AWS) aimed at mitigating food waste by efficiently connecting vendors with surplus food, customers seeking value, and composters for responsible disposal. The overall system architecture is deeply integrated with AWS managed and serverless services. AWS Lambda functions execute core business logic, react to events from DynamoDB Streams and S3 uploads, process queries on behalf of the Bedrock Agent, manage the embedding update lifecycle, and dispatch notifications via Amazon Simple Notification Service (SNS). User identity, authentication, and authorization across different roles (Customer, Vendor, Composter) are securely managed by Amazon Cognito. Amazon S3 provides durable storage for various assets, including user-uploaded images, source data files for AI processing, and the generated embeddings. Geospatial features, such as interactive maps and location-based filtering, are supported by Amazon Location Service. The entire infrastructure stack is defined and managed using the AWS Cloud Development Kit (CDK), promoting consistency, repeatability, and version control through Infrastructure as Code (IaC).

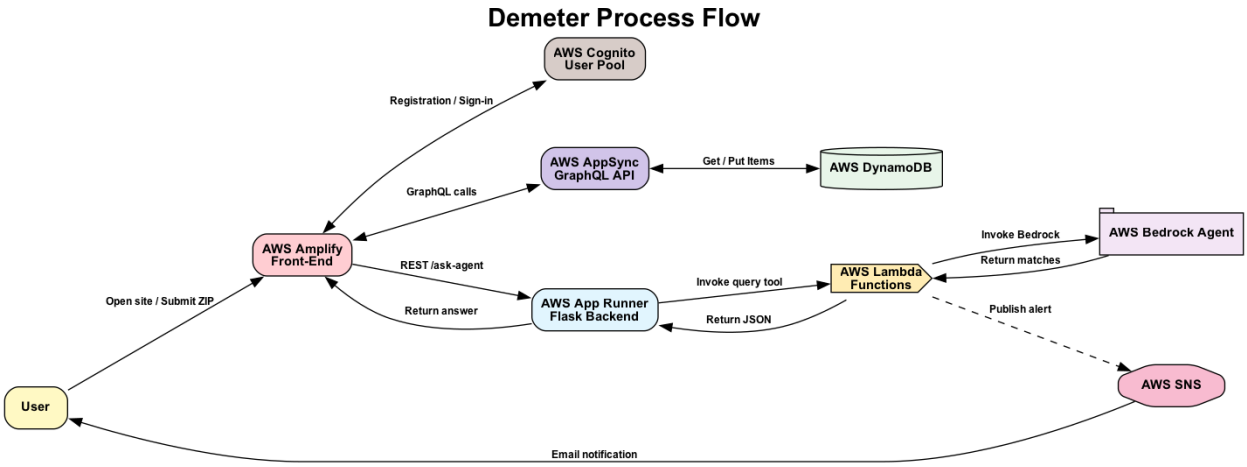
A cornerstone of the user experience is the Demeter Bedrock Agent, an intelligent chatbot powered by Amazon Bedrock. This agent leverages a Retrieval-Augmented Generation (RAG) approach, combining conversational AI reasoning with targeted data lookups to answer user queries based on Demeter's specific datasets, such as locating nearby expiring food items using ZIP codes or addresses. The custom RAG implementation involved a containerized Flask microservice deployed on AWS App Runner, responsible for handling AI-driven search logic and vector embedding workflows using Amazon Bedrock's Titan Embeddings model. This was supported by AWS Lambda functions acting as event triggers and integration bridges, and Amazon S3 serving as the repository for raw data (CSVs) and computed vector embeddings (JSON files).

Looking forward, the Demeter platform is well-positioned to incorporate further enhancements. Potential integrations include leveraging AWS Kendra for broader enterprise search across documentation and support content, implementing advanced conversational bots with Amazon Lex, utilizing Amazon Comprehend for deeper text analytics (sentiment analysis, entity recognition), employing Amazon Rekognition for intelligent image analysis (moderation, object detection, OCR), and developing custom predictive models (e.g., demand forecasting, dynamic pricing, route optimization) using Amazon SageMaker Studio. These future integrations align with the platform's modular, serverless foundation, enabling continuous innovation in pursuit of Demeter's mission to

significantly reduce food waste through technology.

2. Process Flows (Customers, Vendors, Composters)

The Demeter platform facilitates distinct workflows for its three primary user groups—Customers, Vendors, and Composters—orchestrated through its integrated AWS architecture. These flows rely on Amazon Cognito for identity management, AWS AppSync for API interactions, Amazon DynamoDB for data persistence and state management, AWS Lambda for executing business logic and event processing, and Amazon SNS for asynchronous notifications.



2.1. Customer Flow

1. **Registration & Authentication:**
    - Customers register for an account or log in via the frontend web application.
    - Amazon Cognito handles the user registration process, secure authentication (password verification, potentially MFA), and session management.
2. **Browsing & Discovery:**
    - Customers browse available food listings using search filters, categories, or map views on the frontend.
    - The frontend sends GraphQL queries to the AWS AppSync endpoint.

- AppSync resolvers execute these queries against the Demeter_FoodItems DynamoDB table. Efficient retrieval is enabled by leveraging primary keys and Global Secondary Indexes (GSIs) for filtering by attributes like category, vendorId, or sorting by expiryDate.

3. **Conversational Search (via Demeter Bedrock Agent):**
    - Customers interact with the chatbot interface, posing natural language queries.
    - The frontend sends the query to the /ask-agent endpoint hosted on the AWS App Runner service.
    - The service validates the request and invokes the configured Amazon Bedrock Agent using the InvokeAgent API, passing the user's text and session ID.
    - The Bedrock Agent processes the query, identifies intent, and extracts key entities.
    - Based on its configuration, the Agent triggers its designated Lambda function.
    - DemeterQuery Lambda receives the extracted entities and makes an internal HTTP POST request to the /search-data API endpoint on App Runner.
    - The /search-data service performs the RAG retrieval: embeds the query using Amazon Titan, loads pre-computed data embeddings (from S3 JSON in the custom stack), computes vector similarity (e.g., cosine), and returns the top N relevant FoodItem records.
    - The results flow back through DemeterQuery Lambda to the Bedrock Agent.
    - The Agent uses the retrieved data and its internal prompts to synthesize a coherent, conversational response.
    - The response is returned via the /ask-agent service to the frontend chat widget.
4. **Ordering Process:**
    - Customers select items and proceed to checkout via the frontend interface.
    - Placing an order triggers GraphQL mutations via AppSync.The Lambda function executes necessary database operations within DynamoDB, potentially using transactions or conditional writes for atomicity:
        - Creating new records in the Demeter_Orders and Demeter_OrderItems

tables.

        - Decrementing the quantity of the purchased FoodItem in its table.
    - The Lambda may also trigger downstream actions, such as publishing an event to an SNS topic to notify the corresponding Vendor of the new order.

5. **Receiving Notifications (Expiry Alerts):**
    - Customers who have opted-in and configured an alertRadius in their profile can receive proactive notifications.
    - The check_expiring_food_handler Lambda function, triggered by new FoodItem insertions into DynamoDB, checks if the item expires within 24 hours.
    - If an item is nearing expiry, the Lambda scans the Demeter_Users table for customers located within their specified alert radius of the item.
    - For each matching customer, the Lambda publishes a personalized alert message (e.g., "Food item [ID] from vendor [ID] is nearing expiry near your location.") to the designated EXPIRY_ALERT_SNS_TOPIC_ARN.
    - Amazon SNS delivers the message to the customer's subscribed endpoint (e.g., email address).

2.2. Vendor Flow

1. **Registration & Authentication:**
    - Vendors register and log in using the same Amazon Cognito infrastructure as customers, but are assigned the "Vendor" role.
2. **Listing Management:**
    - Vendors use a dedicated section of the platform interface to manage their inventory listings.
    - **Create/Update/Delete:** Actions trigger GraphQL mutations via AppSync, which modify records in the Demeter_FoodItems DynamoDB table. Required attributes include item name, description, category, quantity, price, expiry date, and location.
3. **Order Fulfillment:**
    - Vendors receive notifications about new orders placed for their items. This can

occur via:

- **Real-time Updates:** GraphQL subscriptions via AppSync can push updates directly to the vendor's dashboard.
- **Asynchronous Notifications:** SNS messages (triggered by the order processing Lambda) sent to the vendor's registered email or other endpoints.

- Vendors use the platform interface to view order details and update order statuses (e.g., "Ready for Pickup", "Completed") via AppSync mutations.

4. **Requesting Composting:**
   - For items that are unsold and nearing expiration or are otherwise unsuitable for sale, vendors can mark them as available for pickup by composters.
   - This action typically involves updating the composterAvailability boolean attribute on the FoodItem record in DynamoDB via an AppSync mutation.
   - This update might trigger the match_composters_handler Lambda (if the trigger is on updates) or the Lambda might process it upon initial insertion if the flag is set then.

2.3. Composter Flow

1. **Registration & Authentication:**
   - Composters undergo a specific registration process, providing details relevant to their service, such as:
     - Service Area (defined geographically, perhaps as coordinates and radius, or list of ZIP codes).
     - Accepted Waste Types (e.g., "Produce", "Bakery", "Dairy"). Stored likely as a comma-separated string or list in the acceptedTypes attribute.
     - Certifications or other relevant credentials.
   - Authentication is handled by Amazon Cognito, assigning the "Composter" role. User details are stored in the Demeter_Users DynamoDB table.
2. **Finding Available Items:**
   - Composters use the platform interface to search for food items marked as

available for composting (composterAvailability = true) within their designated service area.

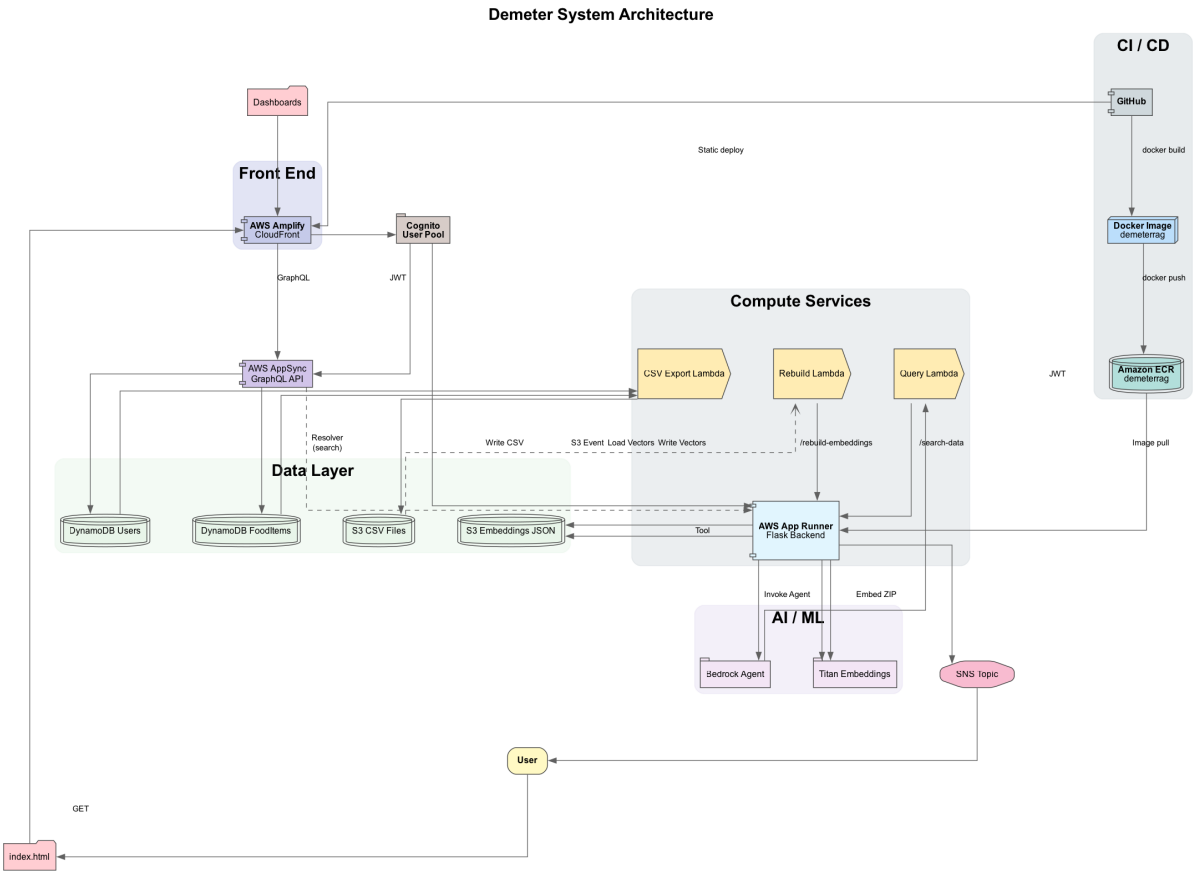
- The frontend sends GraphQL queries via AppSync.
- AppSync resolvers query the Demeter_FoodItems table, filtering by the composterAvailability flag and location. Geospatial filtering uses DynamoDB geohash indexes or calculations based on stored lat/lon coordinates compared against the composter's service area definition.
- Results may also be filtered based on whether the item's category matches the composter's acceptedTypes.

3. **Requesting Pickup:**
   - Composters can select available items and initiate a pickup request via the interface.
   - This triggers AppSync mutations, potentially invoking Lambda functions to:
     - Update the status of the FoodItem (e.g., "Pickup Pending").
     - Create a record in the Demeter_PickupRequests table linking the item, vendor, and composter.
     - Trigger an SNS notification to the vendor informing them of the pickup request.
4. **Receiving Notifications (Composter Matches):**
   - Composters receive proactive alerts about new composting opportunities.
   - The match_composters_handler Lambda function, triggered by new FoodItem insertions marked for composting, scans the Demeter_Users table for composters.
   - It identifies composters whose service area (within the configured COMPOSTER_SEARCH_RADIUS) includes the item's location AND whose acceptedTypes include the item's category.
   - For each matching composter, the Lambda publishes a notification message (e.g., "Vendor [ID] has unsold food (ID: [ID], category: [Category]) near you...") to the COMPOSTER_ALERT_SNS_TOPIC_ARN.

- SNS delivers the alert to the composter's subscribed endpoint.

3. Demeter System Architecture

The Demeter Marketplace operates on a robust, scalable, and modern cloud architecture hosted entirely on AWS. It leverages a serverless-first approach, utilizing managed services to minimize operational overhead and maximize agility and cost-efficiency.



3.1. Core Architectural Principles:

- **Serverless:** Prioritizes the use of services like AWS Lambda, AWS AppSync, Amazon DynamoDB, Amazon S3, and AWS App Runner, which scale automatically and operate on a pay-per-use model, eliminating the need for manual server provisioning and management.

- **Event-Driven:** Components interact asynchronously through events (e.g., S3 object creation triggering Lambda, DynamoDB Streams invoking Lambda, Lambda publishing to SNS), promoting loose coupling and resilience.
- **Managed Services:** Relies heavily on AWS managed services (DynamoDB, Cognito, AppSync, Bedrock, SNS, S3, App Runner, Location Service) to offload undifferentiated heavy lifting like database administration, identity management, API scaling, message queuing, and infrastructure patching.
- **Infrastructure as Code (IaC):** The entire infrastructure is defined, versioned, and deployed using the AWS Cloud Development Kit (CDK), ensuring consistency, repeatability, and automated provisioning via CloudFormation.
- **Scalability & High Availability:** Built on services designed for high availability across multiple AWS Availability Zones (AZs) and automatic scaling to handle fluctuating workloads typical of a marketplace platform.

3.2. Architectural Components:

1. **Frontend Interfaces (Web/Mobile):**
    - **Technology:** Client-side applications likely built with frameworks like React, Vue, Angular, or native mobile technologies (iOS/Android).
    - **Hosting:** Static web assets served via AWS Amplify Hosting or Amazon S3 with Amazon CloudFront distribution for low-latency global delivery and caching.
    - **API Interaction:** Communicates exclusively with the backend via the AWS AppSync GraphQL endpoint, using JWT tokens obtained from Cognito for authentication.
    - **Mapping:** Integrates Amazon Location Service using libraries like MapLibre GL JS to display interactive maps showing food item locations based on latitude/longitude data.
2. **User Authentication (Amazon Cognito):**
    - **Functionality:** Provides user registration, sign-in, password management, MFA, and user profile storage (basic attributes). Manages distinct User Pools or groups for Customers, Vendors, and Composters.
    - **Integration:** Issues JWT tokens used by AppSync for authentication/authorization. Can trigger Lambda functions on certain user lifecycle events (e.g., post-confirmation).

3. **API Layer (AWS AppSync - GraphQL):**
    - **Functionality:** Serves as the single, unified API endpoint for all frontend interactions. Defines the data schema and available operations (queries, mutations, subscriptions) using GraphQL.
    - **Resolvers:** Connect GraphQL operations to backend resources:
        - **Direct DynamoDB Resolvers:** Map simple GraphQL operations directly to DynamoDB actions (GetItem, PutItem, Query, Scan) using Velocity Template Language (VTL) or direct Lambda resolvers.
        - **Lambda Resolvers:** Invoke specific AWS Lambda functions to handle complex business logic, data aggregation across multiple sources, transactional operations, or interactions with other AWS services.
    - **Features:** Supports real-time data updates via GraphQL subscriptions, fine-grained authorization integrated with Cognito, caching capabilities, and automatic scaling.
4. **Serverless Compute (AWS Lambda):**
    - **Runtime:** Executes application code (primarily Python) in response to various triggers without managing servers.
    - **Key Functions:**
        - check_expiring_food_handler: Processes DynamoDB Stream events from Demeter_FoodItems inserts, identifies items expiring within 24 hours, finds nearby customers based on alertRadius, and publishes notifications via SNS.
        - match_composters_handler: Processes DynamoDB Stream events from Demeter_FoodItems inserts (for items marked composterAvailability), finds nearby composters matching service area (COMPOSTER_SEARCH_RADIUS) and acceptedTypes, and publishes notifications via SNS.
        - DemeterQuery: Acts as a Bedrock Agent tool, receiving agent requests, calling the internal /search-data API on App Runner, and returning results.
        - DemeterRebuildEmbeddings: Triggered by S3 CSV uploads, authenticates and calls the /rebuild-embeddings API on App Runner to initiate embedding

updates.

- **Configuration:** Functions are deployed with specific IAM execution roles granting necessary permissions (e.g., DynamoDB read/write, SNS publish, S3 read, Secrets Manager read, permission to be invoked by specific services like AppSync, S3, Bedrock). Environment variables configure dynamic parameters like table names, SNS topic ARNs, App Runner URLs, and search radius.

5. **Data Persistence (Amazon DynamoDB):**
   - **Model:** Fully managed NoSQL key-value and document database. Provides low-latency performance at any scale.
   - **Tables:**
     - Demeter_Users: Stores user profiles keyed by id (UUID). Attributes: role, email, location (Map: lat, lon), alertRadius (Number), acceptedTypes (String). GSIs likely on email (for login lookup) and role.
     - Demeter_FoodItems: Stores listings keyed by id. Attributes: vendorId, category, expiryDate (String - ISO format or YYYY-MM-DD), location (Map: lat, lon), composterAvailability (Boolean). GSIs support queries by vendorId, category+expiryDate, and potentially a geohash GSI for efficient spatial queries.
     - Demeter_Orders, Demeter_OrderItems, Demeter_PickupRequests: Structure data for orders and composting pickups, using appropriate keys and GSIs for access patterns.
   - **Features:** Configured for On-Demand capacity mode for auto-scaling. DynamoDB Streams enabled on Demeter_FoodItems to capture item changes (INSERT events) and trigger notification Lambdas. Supports transactions and conditional writes for data consistency.
6. **File Storage (Amazon S3):**
   - **Usage:** Stores binary objects like user-uploaded images associated with listings. Also serves as the landing zone for raw data CSVs (Demeter_*.csv) and the storage location for generated embeddings (Demeter_*_embedded.json) in the custom RAG stack.
   - **Integration:** Configured with event notifications (on s3:ObjectCreated:* for keys matching Demeter_*.csv) to trigger the DemeterRebuildEmbeddings Lambda.

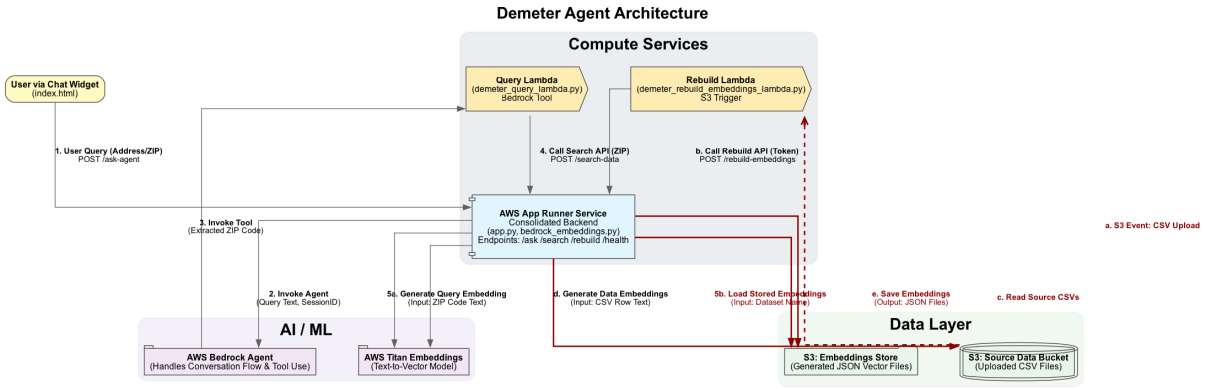
Accessed by App Runner services (read CSVs, write JSONs) and potentially Lambda functions via IAM roles. Can be fronted by CloudFront for optimized image delivery.

7. **Geospatial Services (Amazon Location Service):**
    - **Functionality:** Provides managed APIs for map rendering, geocoding (address -> coordinates), reverse geocoding (coordinates -> address), and place search.
    - **Usage:** Integrated into the frontend for displaying interactive maps with listing markers. Can be used by backend Lambdas or services to convert user-provided addresses into latitude/longitude for storage in DynamoDB, enabling distance calculations (e.g., Haversine formula used in notification Lambdas) and proximity searches.
8. **AI & Search Services:**
    - **Amazon Bedrock:** Provides access to Foundation Models (FMs) like Amazon Titan Text Embeddings for generating semantic vectors. Hosts the conversational Bedrock Agent. Accessed via AWS SDK (InvokeModel, InvokeAgent APIs).
    - **Custom RAG Backend (App Runner/S3/Lambda):** Implements search via vector similarity on embeddings stored in S3 JSON files. (Details in Agent Architecture section).
    - **Managed Alternatives (OpenSearch/Kendra):** Can replace the custom backend, offering managed vector databases (OpenSearch) or intelligent semantic search (Kendra), often integrated via Bedrock Knowledge Bases.
9. **Containerized Microservice (AWS App Runner & Amazon ECR):**
    - **Purpose:** Hosts the containerized Flask backend application, providing REST endpoints needed for the Bedrock Agent integration and embedding management in the custom stack.
    - **ECR:** Stores the built Docker image.
    - **App Runner:** Manages container deployment, scaling (based on configurable metrics), load balancing, health checks, and provides a stable HTTPS endpoint URL. Configured with environment variables (Bedrock IDs, region, secret names) and an IAM instance role granting permissions to call Bedrock, access S3, and potentially read from Secrets Manager.

10. **Notifications & Messaging (Amazon SNS):**
    - **Functionality:** Provides a managed publish-subscribe messaging service for sending asynchronous notifications.
    - **Usage:** Used by Lambda functions (check_expiring_food_handler, match_composters_handler) to send alerts to customers and composters. Specific SNS Topics are configured via environment variables (EXPIRY_ALERT_SNS_TOPIC_ARN, COMPOSTER_ALERT_SNS_TOPIC_ARN).
    - **Benefits:** Decouples notification logic from core business logic, handles message delivery and retries, scales automatically. Supports various endpoints (Email, SMS, Lambda). Custom metrics are logged for monitoring publish success/failure rates.
11. **Infrastructure as Code (AWS CDK):**
    - **Tooling:** Uses AWS CDK with Python to define all AWS resources programmatically.
    - **Process:** CDK code (demeter_stack.py) defines resources like DynamoDB tables, AppSync API, Lambda functions (including code packaging), App Runner service, ECR repository lookup, S3 bucket configurations (notifications), SNS Topics, IAM roles/policies, and Secrets Manager integration. The cdk deploy command synthesizes and deploys a CloudFormation stack.
    - **Benefits:** Ensures infrastructure consistency, enables version control, automates provisioning, simplifies environment replication, and manages dependencies between resources. Uses context parameters (-c) for deployment flexibility.

4. Demeter Bedrock Agent Architecture

The Demeter Agent is a specialized component designed to provide an intelligent, conversational interface for users interacting with the Demeter Marketplace data. It utilizes Amazon Bedrock's capabilities for natural language understanding, reasoning, and integration with external data sources through a Retrieval-Augmented Generation (RAG) architecture. This section details the architecture, primarily focusing on the initial custom stack implementation described in the source documents, while also outlining the recommended simplification using Bedrock Knowledge Bases.



4.1. Core Principles:

Retrieval-Augmented Generation (RAG): Enhances the agent's responses by retrieving relevant, up-to-date information from a custom knowledge base (Demeter's data) before generating the final answer. This ensures responses are grounded in specific data (e.g., actual food listings) rather than just the FM's general knowledge.

- **Modular Design:** Components like the agent, data retrieval logic, and embedding management are separated into distinct services (Bedrock Agent, Lambda functions, App Runner services), interacting via APIs and events.

4.2. Architectural Components (Custom Stack Implementation):

1. **User Interface (Frontend):**
    - A chat widget embedded in the Demeter web or mobile application.
    - Captures user input (text queries).
    - Sends HTTP POST requests containing the query and a session ID to the /ask-agent backend endpoint.
    - Receives and displays the agent's final response.
2. **Backend API Gateway (bedrock-agent-service on App Runner):**
    - **Endpoint:** Hosts the public-facing /ask-agent REST endpoint.
    - **Function:** Acts as a proxy between the frontend and the Amazon Bedrock Agent service.

- **Logic:**
  - Receives POST request with question and optional sessionId.
  - Validates input and manages sessionId (reuses or generates new UUID).
  - Uses AWS SDK (boto3) to call bedrock-runtime.invoke_agent.
  - Provides agentId, agentAliasId (from environment variables BEDROCK_AGENT_ID, BEDROCK_AGENT_ALIAS_ID), sessionId, and inputText.
  - Optionally enables tracing (enableTrace=False).
  - Handles the response stream from invoke_agent, decoding and concatenating chunk bytes into the final answer string.
  - Returns a JSON response {"answer": final_answer} or {"error": ...}.
- **Permissions:** Requires IAM bedrock:InvokeAgent permission on the specific agent ARN.

3. **Amazon Bedrock Agent:**
   - **Configuration:** Defined in the Bedrock console or via API. Configured with:
     - **Foundation Model:** Selects the underlying FM for reasoning and generation.
     - **Instructions (Prompt Template):** Defines the agent's persona, capabilities, constraints, and how it should process input, use tools, and format output. Crucial for guiding the RAG process (e.g., "If the user asks about food items near a location, extract the ZIP code and use the DemeterQuery tool...").
     - **Action Groups/Tools:** Defines interfaces to external systems. In this custom stack, it includes a Lambda tool configured to invoke the DemeterQuery function. The tool definition specifies the Lambda ARN, input schema (e.g., expecting a zipCode parameter), and output schema (e.g., expecting a matches field).
     - **IAM Role:** An execution role that the Bedrock service assumes when invoking tools (Lambdas). This role must grant lambda:InvokeFunction permission for the DemeterQuery Lambda ARN.

- **Processing Flow:**
  - Receives input text and session context from invoke_agent.
  - Applies instructions to understand intent and extract parameters (e.g., ZIP code).
  - Determines the need to use the DemeterQuery tool.
  - Invokes the DemeterQuery Lambda, passing the extracted parameters (e.g., {"zipCode": "67890"}).
  - Receives the JSON response (containing matches) from the Lambda.
  - Incorporates the retrieved matches data into its context.
  - Uses its instructions and the underlying FM to generate a final, contextually relevant, natural language response based on the retrieved data.
  - Returns the response stream to the invoke_agent caller.

4. **Agent Tool (DemeterQuery Lambda):**
   - **Purpose:** Acts as an intermediary, translating the Bedrock Agent's tool invocation into an internal API call.
   - **Trigger:** Synchronous invocation by the Bedrock Agent service.
   - **Logic:**
     - Receives event payload (e.g., {"zipCode": "..."}).
     - Constructs an HTTP POST request to the internal /search-data endpoint (URL from APP_RUNNER_URL env var).
     - Sends the query data in the request body (e.g., {"question": zip_code, "dataset": "FoodItems"}).
     - Waits for the response from the /search-data service.
     - Returns the received JSON response directly to the Bedrock Agent.
   - **Permissions:** Needs network access to call the App Runner endpoint. The Bedrock Agent's execution role needs permission to invoke this Lambda.

5. **Data Retrieval Service (demeter-rag-service / /search-data on App Runner):**
    - **Endpoint:** Hosts the internal /search-data REST endpoint.
    - **Purpose:** Performs the core vector similarity search (the "Retrieval" in RAG).
    - **Logic:**
        - Receives POST request from DemeterQuery Lambda containing the query text (e.g., ZIP code).
        - **Query Embedding:** Calls Bedrock InvokeModel API using the configured Titan Embeddings model ID (amazon.titan-embed-text-v2:0) to get the query vector.
        - **Load Data Embeddings:** Reads the corresponding Demeter_<Dataset>_embedded.json file from S3 into memory. This file contains { "embedding": [...], "item": {...} } objects.
        - **Similarity Calculation:** Computes cosine similarity between the query vector and each item's embedding vector using libraries like numpy.
        - **Ranking & Selection:** Sorts items by descending similarity score and returns the top N matches.
        - **Response:** Returns JSON {"matches": [top_item_1, top_item_2, ...]}.
    - **Permissions:** Requires IAM bedrock:InvokeModel (for Titan) and s3:GetObject (for embeddings JSON).
6. **Embedding Generation Service (demeter-lambda-service / /rebuild-embeddings on App Runner):**
    - **Endpoint:** Hosts the internal /rebuild-embeddings REST endpoint.
    - **Purpose:** Regenerates the embeddings stored in S3 JSON files.
    - **Trigger:** Called by the DemeterRebuildEmbeddings Lambda.
    - **Authentication:** Requires a valid X-Admin-Token header matching the configured ADMIN_TOKEN environment variable (sourced from Secrets Manager).
    - **Logic:** Executes the embedding generation script (e.g.,

tfidf_embeddings.run_all() or bedrock_embeddings.py):

- Reads source data CSVs (Demeter_*.csv) from S3.
- For each record, generates an embedding vector (using TF-IDF locally or by calling Bedrock Titan InvokeModel).
- Constructs the JSON structure containing embeddings and item metadata.
- Writes/overwrites the Demeter_<Dataset>_embedded.json file in S3.

- **Permissions:** Requires IAM s3:GetObject (for CSVs), s3:PutObject (for JSONs), and potentially bedrock:InvokeModel if using Titan for data embedding. Also needs permission to read the admin token from Secrets Manager if configured that way.

7. **Embedding Update Trigger (DemeterRebuildEmbeddings Lambda):**
   - **Trigger:** S3 ObjectCreated events for Demeter_*.csv files.
   - **Logic:** Parses event, verifies key pattern. Makes authenticated POST request to /rebuild-embeddings endpoint, passing the ADMIN_TOKEN (from env var) in the header.
   - **Permissions:** Needs network access to App Runner. S3 bucket needs permission to invoke this Lambda. Lambda needs permission to read ADMIN_TOKEN env var (potentially sourced from Secrets Manager).
8. **Data & Embedding Storage (Amazon S3):**
   - Stores source Demeter_*.csv files.
   - Stores generated Demeter_*_embedded.json files containing vectors and metadata.
9. **Embedding Model (Amazon Titan Text Embeddings):**
   - Accessed via Bedrock InvokeModel API.
   - Used by /search-data to embed user queries.

5. Future Enhancements

The Demeter Marketplace platform, with its current serverless and AI-integrated architecture, provides a strong foundation. However, several potential future enhancements leveraging additional AWS services can further augment its capabilities, improve user experience, optimize operations, and deepen its impact on reducing food waste.

1. **Demeter Bedrock Agent Knowledge Base Enhancement**

Choosing the optimal backend technology for the Demeter Bedrock Agent's knowledge base involves evaluating trade-offs between cost, complexity, search capabilities, integration effort, scalability, and maintenance. The initial custom stack (AWS App Runner, Lambda, S3 Embeddings) encountered operational difficulties, prompting a detailed comparison with managed AWS alternatives: Amazon OpenSearch Serverless (configured for vector search) and Amazon Kendra (an intelligent search service with Developer and GenAI Editions).

Comparative Analysis:

Feature/Aspect	Current Custom Stack (App Runner/Lambda/S3)	Amazon OpenSearch Serverless (Vector Search)	Amazon Kendra (Developer / GenAI Editions)
Core Technology	Custom Python App (Flask), Lambda Functions, S3 JSON Files	Managed OpenSearch Service (Vector Database)	Managed Intelligent Search Service (ML/ Deep Learning Models)
Data Handling/ Indexing	**Manual Pipeline:** Custom scripts for parsing (limited formats), embedding (via Bedrock), JSON formatting, S3 storage. **High Effort.**	**Semi-Managed:** Requires external pipeline for embedding generation & data ingestion into OpenSearch. OpenSearch manages index infra. **Medium Effort.**	**Fully Managed:** Built-in connectors, auto-parsing (PDF, DOCX, HTML+), auto-chunking, managed indexing (can use Bedrock embeddings via KB). **Low Effort.**
Retrieval Mechanism	**Basic Vector Search:** Custom code loads S3 JSON, embeds query, computes cosine similarity.	**Optimized Vector Search:** Uses k-NN/ ANN algorithms (Faiss/ NMSLIB) via OpenSearch API. Supports metadata filtering.	**Intelligent Semantic Search:** Uses ML models for NL understanding, context, synonyms. Retrieve API optimized for RAG. Features: Direct Answers, Highlighting, Tuning.

Search Sophistication	**Low:** Basic semantic similarity based on embeddings. Lacks NL understanding, relevance control.	**Medium:** Efficient vector similarity. Good for finding "like" documents. Lacks advanced QA features. Quality depends on external embedding model.	**High:** Designed for QA & discovery. Understands intent, provides relevant passages, tunable relevance. Superior for RAG use cases.
Bedrock Integration	**High Complexity:** Requires custom Action Group -> Intermediary Lambda -> App Runner API. Developer manages entire flow.	**Medium Complexity (Rec: KB):** Bedrock Knowledge Base abstracts query/ retrieval. **High Complexity (Custom):** Action Group -> Lambda -> OpenSearch API.	**Low Complexity (Rec: KB):** Bedrock Knowledge Base natively integrates, handles query/retrieval via Kendra API. **Medium Complexity (Custom):** Action Group -> Lambda -> Kendra API.
Scalability	**Manual/Semi-Manual:** Requires tuning App Runner scaling, Lambda concurrency. Custom search code can bottleneck.	**Automatic:** Managed scaling of OpenSearch Compute Units (OCUs) and storage by AWS.	**Automatic:** Managed scaling based on Kendra Edition (Developer/GenAI) and usage, handled by AWS.
Maintenance Effort	**High:** Maintain custom code (Flask, Lambdas, scripts), dependencies, infrastructure config, debug integration issues.	**Medium:** Maintain embedding/ingestion pipeline code & dependencies, monitor OCU costs, potentially tune index mappings.	**Low:** Configure data sources/syncs, monitor index health, optionally perform relevance tuning. AWS manages infra/software updates.
Cost Profile (Excl. Bedrock)	Low Infra Cost (if within ongoing free tiers) + **High Hidden Maintenance Cost**.	Medium (OCU hours + Storage GB-months) + **Cost of Embedding Pipeline**.	Higher Base Cost (Hourly rate for Index) + Usage Overage + **Low Maintenance Cost**. GenAI Edition significantly more expensive hourly than Developer.
AWS Free Tier	**Ongoing Monthly:** App Runner compute/ memory, Lambda requests/duration, initial S3.	**Time-Limited:** Typically first month only for OCUs & storage.	**Time-Limited:** Typically first 30 days only for index hours.

2. **Advanced Conversational Interfaces (Amazon Lex):**
    - **Concept:** Implement more sophisticated, goal-oriented conversational bots for tasks beyond simple Q&A or data retrieval.
    - **Technology:** Utilize Amazon Lex V2 to design and build multi-turn conversational flows.
    - **Benefits:** Create guided experiences for users (e.g., a "Listing Assistant" for vendors, an "Order Placement Bot" for customers, a "Pickup Scheduler" for composters). Lex handles intent recognition, slot filling (collecting required information), dialogue state management, and supports both text and voice interactions. Can integrate seamlessly with backend logic via Lambda functions.
    - **Implementation:** Design conversational flows, define intents and slots using the Lex console or APIs. Integrate the Lex bot into the frontend using AWS SDKs. Trigger Lambda functions from Lex intents to execute actions (e.g., call AppSync APIs, update DynamoDB).
3. **Deeper Text Analytics (Amazon Comprehend):**
    - **Concept:** Apply advanced Natural Language Processing (NLP) techniques to extract richer insights and automate tasks based on textual data generated within the platform (listings, reviews, queries).
    - **Technology:** Utilize Amazon Comprehend's managed NLP services.
    - **Benefits:**
        - *Enhanced Categorization:* Use custom classification to automatically categorize listings based on descriptions, improving consistency.
        - *Sentiment Analysis:* Analyze customer feedback or reviews to gauge overall satisfaction, identify recurring issues, or flag problematic interactions.
        - *Trend Analysis:* Apply topic modeling to user search queries or support interactions to understand user needs and inform feature development.
    - **Implementation:** Invoke Comprehend APIs (e.g., DetectSentiment, DetectEntities, ClassifyDocument) from Lambda functions triggered by relevant events (e.g., new listing creation, review submission).

4. **Intelligent Image Analysis (Amazon Rekognition):**
    - **Concept:** Leverage computer vision to analyze images uploaded by vendors, enhancing data quality and automation.
    - **Technology:** Utilize Amazon Rekognition's image analysis capabilities.
    - **Benefits:**
        - *Object & Scene Detection:* Identify objects within food photos (e.g., "apple", "bread", "packaged meal") to verify listing details or suggest tags/ categories.
        - *Text Recognition (OCR):* Extract text from images of packaging or labels (e.g., expiration dates, nutritional information, brand names) to reduce manual data entry for vendors.
    - **Implementation:** Trigger a Lambda function when new images are uploaded to the S3 bucket. The Lambda calls Rekognition APIs (DetectModerationLabels, DetectLabels, DetectText). Analysis results can be used to flag listings for review, update DynamoDB records, or provide suggestions to the vendor.

By strategically incorporating these enhancements, the Demeter Marketplace can continuously evolve, offering a more intelligent, efficient, and user-friendly platform that further strengthens its mission to combat food waste. The modular, AWS-native architecture facilitates the integration of these advanced services with relatively low friction.

Appendix

main.dgchwyhaw9222.amplifyapp.com/index.html

DEMETER

Home Customers Vendors Composters Register Sign In

Reshape Food Waste Into Purpose

At Demeter, we transform surplus into service—empowering communities to reclaim food, restore balance, and reduce waste.

Rooted in Community

Demeter connects local vendors, composting partners, and conscious consumers to build a sustainable food network. We believe nourishment should reach people—not landfills.

Sign-in to Explore the Marketplace

Demeter Agent

Find Expiring Food Items Nearby

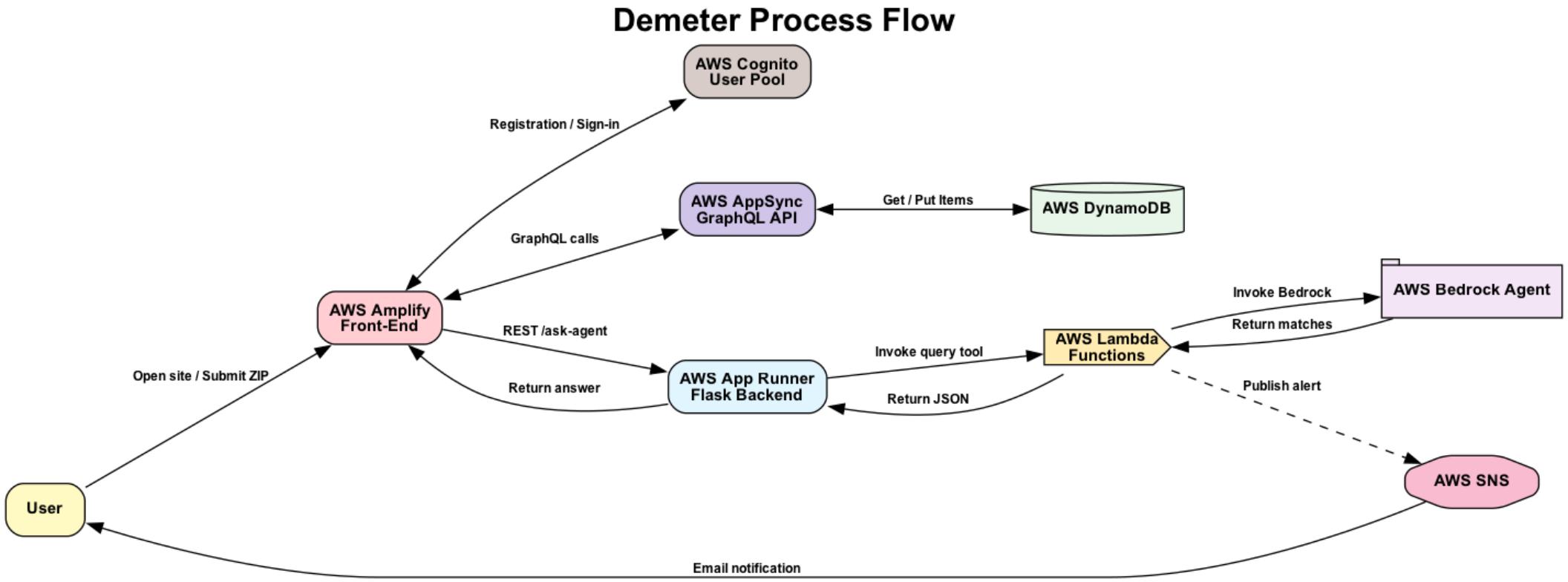
I can help you find expiring food items nearby. Please provide me with your address including the ZIP code.

3622 Wilkens Ave Baltimore MD 21228

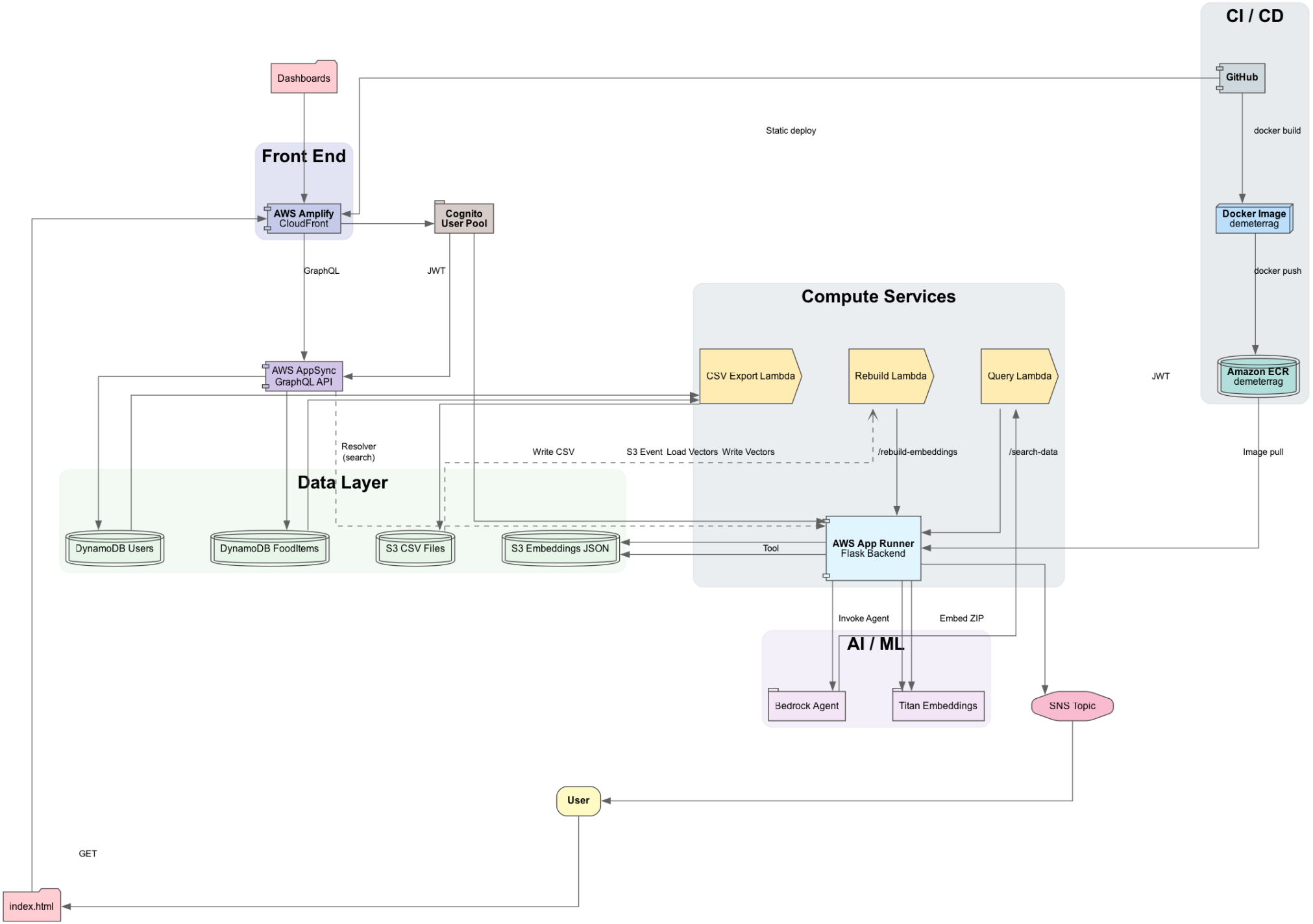
Thank you. Let me check for expiring food items in your area.

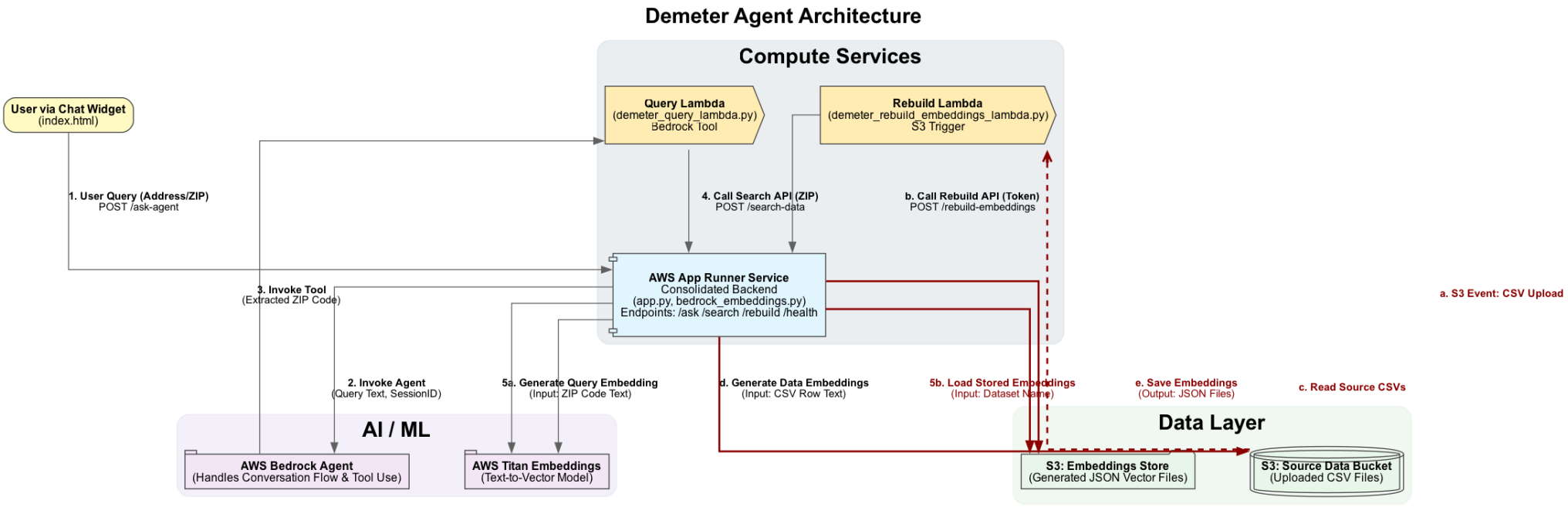
Based on your address, the ZIP code is 21228. Here are three food items from the knowledge base based on the earliest expiry date whose ZIP code matches your ZIP code: 1.

Ask something...


Demeter Process Flow
AWS Cognito User Pool
Registration / Sign-in
AWS AppSync GraphQL API
Get / Put Items
AWS DynamoDB
GraphQL calls
AWS Amplify Front-End
Invoke Bedrock
AWS Bedrock Agent
Return matches
REST /ask-agent
Invoke query tool
AWS Lambda Functions
Open site / Submit ZIP
Return answer
AWS App Runner Flask Backend
Return JSON
Publish alert
AWS SNS
User
Email notification

Demeter System Architecture




Demeter Agent Architecture
Compute Services
User via Chat Widget
(index.html)
Query Lambda
(demeter_query_lambda.py)
Bedrock Tool
Rebuild Lambda
(demeter_rebuild_embeddings_lambda.py)
S3 Trigger
1. User Query (Address/ZIP)
POST /ask-agent
4. Call Search API (ZIP)
POST /search-data
b. Call Rebuild API (Token)
POST /rebuild-embeddings
AWS App Runner Service
Consolidated Backend
(app.py, bedrock_embeddings.py)
Endpoints: /ask /search /rebuild /health
3. Invoke Tool
(Extracted ZIP Code)
a. S3 Event: CSV Upload
2. Invoke Agent
(Query Text, SessionID)
5a. Generate Query Embedding
(Input: ZIP Code Text)
d. Generate Data Embeddings
(Input: CSV Row Text)
5b. Load Stored Embeddings
(Input: Dataset Name)
e. Save Embeddings
(Output: JSON Files)
c. Read Source CSVs
AI / ML
Data Layer
AWS Bedrock Agent
(Handles Conversation Flow & Tool Use)
AWS Titan Embeddings
(Text-to-Vector Model)
S3: Embeddings Store
(Generated JSON Vector Files)
S3: Source Data Bucket
(Uploaded CSV Files)