Deployment Architecture for Solar Plant Detection System

1. System Overview

The Solar Plant Detection System is designed to identify solar power stations from satellite imagery using a machine learning-based pipeline. The system consists of several AWS-managed services, including Lambda Functions, AWS Batch Jobs, API Gateway, Step Functions, and S3 storage, to ensure scalability, automation, and robustness.

The deployment architecture supports image enhancement, prediction, and report generation, orchestrated through AWS Step Functions and Batch Jobs.

---

2. Major Components and Data Flow

a) Inputs & Outputs

- **Inputs**: Geographic coordinates (center point), region size (NS & WE distance in km).
- **Outputs**: Processed images, statistical reports, and downloadable results in a ZIP file.

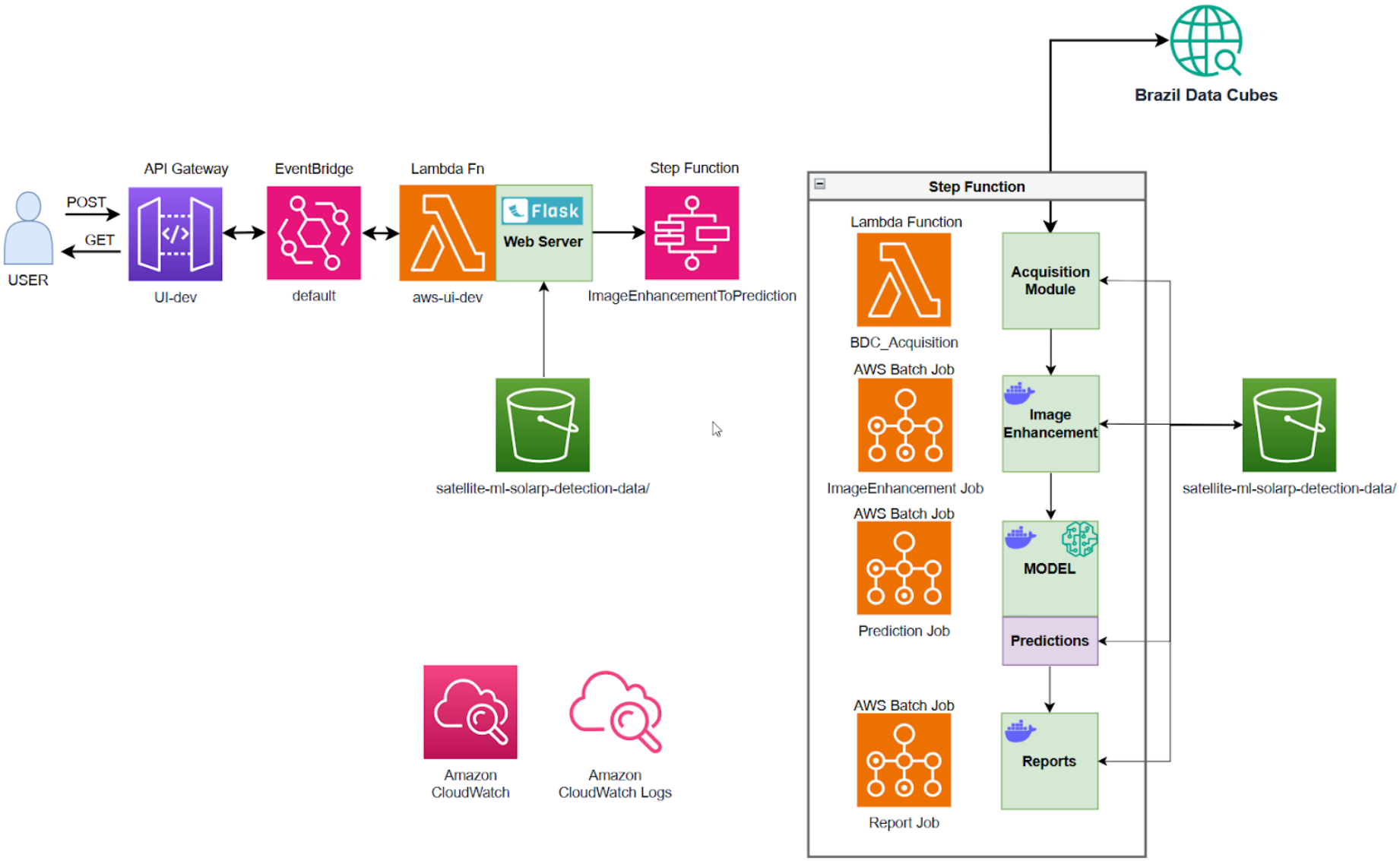
b) Major Components

1. **User Interface (Web Server)**
    - Allows users to input detection parameters and trigger analysis.
    - Hosted on AWS Lambda and served via API Gateway.

2. **API Gateway**
    - Facilitates communication between the user interface and backend services.

3. **Step Function (ImageEnhancementToPrediction)**
    - Orchestrates the ML pipeline, managing data flow between image processing, prediction, and report generation.

Design of the Deployment Solution Architecture:



4. **AWS Lambda Functions**

    - Handles event-driven automation, pre-processing, and post-processing.
    - Generates signed URLs for secure file access.

5. **AWS Batch Jobs**

    - **Image Enhancement Job**: Prepares satellite imagery before ML model inference. Upscales images with Bicubic interpolation.
    - **Prediction Job**: Runs the ML model for solar panel detection.
    - **Report Job**: Generates statistical analysis and reports.

6. **Amazon S3 (satellite-ml-solarp-detection-data/)**

    - Stores raw images, processed images, predictions, and reports.

7. **Amazon CloudWatch**

    - Monitors logs and system performance for debugging and optimization.

---

3. Transaction Management

- The workflow follows a simple yet robust transaction identification system to track inputs and outputs across modules.
- Each request sent to the system(main inputs) is assigned to a unique ***transaction_id,*** generated by the BCD_Acquisition Module. The transaction ID follows this structure:

    NNNNNN-YYYY-MM-DD:

    NNNNNN: a six-digit incremental counter

    YYYY-MM-DD: the date of the transaction

- The ***transaction_id*** is used throughout the workflow to identify and organize data. Each module stores its outputs in a dedicated folder structured as: ***./'module_name'/'transaction_id'***.
- This approach ensures that each module can efficiently retrieve and store intermediate images, facilitating a seamless workflow operation.

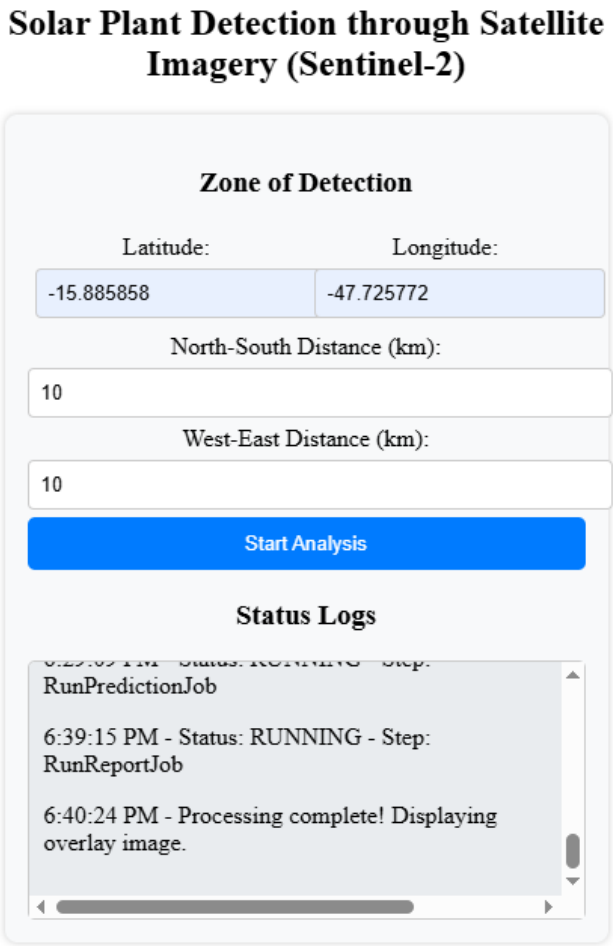
3. Data Storage and Management

- Raw satellite imagery is sourced from Brazil Data Cube (BDC).
- Processed images and predictions are stored in an Amazon S3 bucket (`satellite-ml-solarp-detection-data/`).
- Reports are generated and stored within S3 under the `/reports/{transaction_id}/` path.
- Execution logs are stored in CloudWatch for monitoring and debugging.

---

4. Data Flow Between Components

1. User submits a request via the web interface (latitude, longitude, region size).



2. API Gateway triggers aws-ui Lambda, which starts the Step Function.
3. Step Function orchestrates:

- BCD_Acquisition (Lambda) and the consecutive AWS Batch jobs:
- Image Enhancement → Prediction → Report Generation.

4. Prediction results and reports are stored in S3.
5. aws-ui Lambda Function generates a pre-signed URL for downloading results.
6. User downloads results via the web interface.

Satellite Report for 000176-2025-03-17

Satellite imagery and detection mosaic overlay

Statistics

Cell	Total Pixels	Total Area (km^2)	Positive Pixels	Positive Area (m^2)	Coverage %
1	262,144	6.554	0	0	0.000000
2	262,144	6.554	0	0	0.000000
3	262,144	6.554	0	0	0.000000
4	262,144	6.554	0	0	0.000000
5	262,144	6.554	0	0	0.000000
6	262,144	6.554	0	0	0.000000
7	262,144	6.554	0	0	0.000000
8	262,144	6.554	0	0	0.000000
9	262,144	6.554	0	0	0.000000
10	262,144	6.554	0	0	0.000000
11	262,144	6.554	4,025	100,625	1.535416
12	262,144	6.554	0	0	0.000000
13	262,144	6.554	0	0	0.000000
14	262,144	6.554	0	0	0.000000
15	262,144	6.554	14	350	0.005341
16	262,144	6.554	0	0	0.000000

5. ML Model Lifecycle

a) Model Retraining Strategy

The retraining was considered as part of the system design. Each one of the sub-images that compose the detection or inference labels can be easily identified by the overlay grid and the corresponding image of each of the internal steps are available in the S3 bucket. For example: if we identified sub-images with False Positives and no False Negatives, we can identify the input image that belongs with the specific grid and associate it with a all-black mask (all negative pixels) and use it for the retraining. In case that we detect, False Negatives, True positives that were not detected by the Model, we can also identify the input image and then prepare the corresponding binary mask image and also retrain the model.

From our experience this model is more susceptible to False Positives, then the retraining procedure should be simpler than with False Negatives.

- **Retraining frequency**:
  - Performance metrics degrade below a predefined threshold. We found that a threshold of 0.90 keeps a good balance for detection.
  - New labeled data becomes available.
- **Retraining data storage**:
  - New labeled images and prediction results are stored in **Amazon S3**.
- **Evaluation**:
  - Performance can be validated using new test datasets.
- **Deployment**:
  - Updated model versions are deployed to AWS Batch Jobs.

---

6. System Monitoring & Debugging

- **Amazon CloudWatch**:
  - Captures logs for Lambda functions and Batch Jobs.
  - Alerts are set for failures in Step Functions.
- **API Gateway Logs**:
  - Tracks user interactions and API requests.
- **Pre-signed URL logging**:
  - Ensures proper S3 permissions for downloads.

7. Technologies Used

Component	Technology
Frontend UI	HTML, JavaScript (fetch API)
Backend API	Flask (Python) and Zappa for deployment
Cloud Infrastructure	AWS Lambda, API Gateway
Orchestration	Step Functions
Data Storage	Amazon S3
Batch Processing	AWS Batch, Docker containers
Monitoring	Amazon CloudWatch
ML Model	U-Net (Convolutional Deep Learning model)

---

8. Scalability, Fault Tolerance & Performance Considerations

- **Scalability**:
  - AWS Lambda scales automatically based on requests.
  - AWS Batch handles parallel job execution for large image processing tasks.
- **Fault Tolerance**:
  - Step Functions ensure error handling and retries for failed jobs.
  - CloudWatch logs provide real-time failure monitoring.
- **Performance Optimization**:
  - Increased Lambda memory allocation for handling large images.
  - Pre-signed S3 URLs reduce data transfer overhead.

9. Estimated Deployment Cost

Considering 30 transactions per day with an average of 20 sub-images.

Service	Estimated Cost (per month)
AWS Lambda (API & Reports)	$5 - $10
API Gateway	$2 - $5
Step Functions	$5 - $10
AWS Batch Jobs	$15 - $35
Amazon S3 Storage	$5 - $10
CloudWatch Monitoring	$1 - $3
Total Estimate*	**$33 - $73**

** based on usage.*

---

10. Handling Edge Cases

Scenario	Solution
Large Images causing download failures	Increase Lambda memory, move to direct S3 downloads via pre-signed URLs.
Step Function failure	Implement retry policies, log errors in CloudWatch.
API timeout due to large processing	Move long-running tasks to AWS Batch for better resource allocation.

---

11. Conclusion

This deployment architecture ensures an efficient, scalable, and fault-tolerant solution for detecting solar power stations in satellite imagery. AWS Lambda, Step Functions, and Batch Jobs provide seamless orchestration, while S3 and CloudWatch enable data management and monitoring. The system is designed for automatic scaling, optimized performance, and cost-effectiveness. The API contemplates the possibility of integration with other systems, not only the use of a frontend.

ANNEX I: API Gateway Definitions

Following, we present the routes, methods and flask functions published through the API Gateway.

/

GET: home(): brings the home page.

/start

POST: start_workflow():

API point of entry. Initiates the workflow, calls the Step Function.

/status

GET: check_status():

Retrieves the transaction logs and brings information about the workflow step name and state.

/get-report

GET: get-report():

Generates and pre-signed URL for the report.

/download-results

GET: download_results()

Creates and serves a ZIP file with results, including:
input.png: input images in a mosaicking presentation
prediction.png: inference image labels in a mosaicking presentation
overlay.png: overlay the input and prediction images
report.html: html report with the detection main results