Technical Assignment – Project Egrid Demo

Architecture Impact Assessment:

Project Description: To design and implement a solution that enables visualization of the annual net generation of U.S power plants.

Core Requirements:

1. Ingest CSV files from object storage
2. Transform, store and expose the data
3. Build a simple UI
4. Containerize the solution
5. Document the work.

Architecture Options & Recommendation:

Option	Overview of Solution	Dependencies	Architecture Assessment
1	Localsetup using containers: **Frontend (Nginx container)** • Based on nginx: alpine (Linux image). • Serves the static index.html + JS files. **API (FastAPI container)** • Python + FastAPI app, runs with uvicorn. • Exposes endpoints /top and /search. **DynamoDB Local container** • Based on amazon/DynamoDB-local: latest. • Emulates AWS DynamoDB **MinIO container** • Based on minio/minio: latest. • Provides an S3-compatible storage service **Ingest container**	-Software: Docker compose, Html, MinIO, Nginx -Technical Skills -Coding complexity -No performance test model available to guarantee on scalability - Availability and Resilience are not proven	Easy for testing on local environment, cost effective

	- Python worker app (custom code). - Polls MinIO for new CSV files, parses them, and writes to DynamoDB.		
2	Using Managed Services in AWS **Services in AWS Setup** **Frontend (CloudFront + S3 Site Bucket)** - CloudFront serves the static frontend (index.html + JS). - Backed by an S3 bucket that stores the website assets.	AWS Account / Environment, -Terraform -Cost	Very easy to build and maintain, auto scalable and resilient. -Perfect match for requirements.
	API (Lambda + API Gateway) - Lambda function (egrid-api) runs the FastAPI-equivalent logic. - Invoked via **API Gateway (HTTP API)**. - Exposes endpoints /top and /search over HTTPS.		
	Database (DynamoDB Managed Service) - Fully managed DynamoDB table (egrid_plants). - Stores normalized data		
	Object Storage (Amazon S3 Data Bucket) - S3 bucket stores incoming CSV files - Fully S3-compatible, accessible via HTTPS API. - Triggers ingestion Lambda on new file upload.		

3	Using **Powerdrill AI** tool: No code, AI powered data analysis platform designed to help users extract insights from data. Support Natural language queries and generates visualizations and produce reports -Can work directly from uploads, minimal engineering - very fast to get attractive charts from CSV's.	- Cannot restrict on the input file type. -Limitations of free version -Capacity of free version is 5MB only	Saas tool, quick time to market. -However, prevalidation logic has to be integrated to restrict the input file type to only .CSV. -Doesn't fit the core requirements also

Scope: Limited only to the technical assignment – Demo purpose.

Proposed Solution: Both Option 1 & Option 2

To create two solutions one for local testing and other solution using Managed services on AWS to address all the core requirements.

Non-Functional Requirements:

Attribute	Requirement
Highest Data Classification Public/Internal/ Restricted/Highly Restricted	Public
Service Tier (0/1/2)	N/A
Maximum Disruption Time RTO+ 1hr Max	N/A
RTO (HH:MM: SS)	N/A
RPO (HH:MM: SS) Point of Failure Near Point of Failure Last daily backup	Last Backup
Service Hours (OLA's) /Time Zone	Core Hour/Peak Hour: Online Hour: Maintenance Window: Restricted Window: Batch Window:

Service Performance Users Capacity Growth Response Times etc	Latency Targets: 2 seconds load time for UI screen Volume / Throughput: XX TPS on staff channel XY TPS on Browser channel XZ TPS on Mobile channel Table added below*
Consuming Regions, types and location of users	Region: INDIA User Location: ABUDHABI
Regulatory Requirements /Other Requirements: Incountry Data Hosting/Cloud Hosting /Compliance /Audit findings etc	N/A

*

Metric	Day1	End of yr1	End of yr2	End of yr3	End of yr4	End of yr5
Internal Staff Users						
Customer: Web/Mobile						
External Users (3rd party)						
Concurrent Users						
Total Users						
Peak User volumes /Day						
Total Transactions / Day						
Response Time Targets						
Batch Frequency /Day						
Batch File Size						

Risks & Issues:

Phase	Area	Summary	Description	Inherent Risk	Residual Risk
AWS Setup	Security	No VPC is created	All AWS server less services are	High	Low

			publicly accessible		
AWS	Access Controls	No Authentication	No Auth service is integrated	High	Low
AWS	Availability	Single Region, Single AZ setup	If region goes down, setup will be unavailable	High	Low
Local setup	Frontend	The protocol used is Http	Since this is the Demo setup, the protocol used at the browser is Http instead of Https	High	Low
Local Setup	Resiliency	No Resiliency	Since this is demo setup, there is no high availability consideration	Low	Low
Document store	Public Repo	Used Public GitHub for storing docs	The project docs are uploaded into Public Git	Low	Low
Local Setup	Logging Tracing Monitoring, Alerting	No Agent integration	No automated setup	High	Low

Key Design Decisions:

Phase	Area	Summary	Impact & Rationale

Design	Two-Pronged approach for Design	Split the design and implementation into local setup and Cloud Setup	Local Setup to use containers and perform easy testing. AWS setup to address Scalability, Resilience and handle changing NFR's
Implementation	Basic, simplistic model	Very basic setup has been designed for implementation considering the time and tools/resource /environment availability	Just a demo project, cannot be refered.

Technology Stack:

Local Setup Components & Rationale:

S. No	Local Component	Purpose	Justification	Comments
1	Frontend	Nginx serving the static UI	Static HTML single page is rendered	Nginx Alpine Image, light weight
2	MinIO	Storage Bucket	S3-compatible store for the CSVs uploads	Raw Data Store
3	Dynamo DB	Database	Local DynamoDB so we can test without AWS	Normalized Data lives here
4	Ingest	ETL poller	Watches MinIO and	Extracts, cleans data

			Polls as soon as files upload	and writes to DB
5	API	FastAPI to integrate FE with Database	Serves the frontend request by fetching the data from DB	Sort, Filter, Fetch data from DB

AWS Setup Components & Rationale:

S. No	AWS Component	Purpose	Justification	Comments
1	S3	Holds the raw CSV uploads	Durable, cheap, event-driven; simplest place to drop files	Free tier + storage events are free
2	Lambda (ingest)	Parses CSV, cleans, aggregates, loads to DynamoDB	Serverless, auto-scales, no servers to manage	pay per ms;
3	DynamoDB	Stores clean plant records	Serverless, pay-per-request, great for simple reads	Free tier; on-demand pricing keeps cost low
4	Lambda (api)	Reads DynamoDB and returns JSON	Same serverless benefits; tiny code surface	Since less invocations, might hit cold start and cost more.
5	API Gateway	Public HTTPS endpoints	Managed auth/throttling/logging/Rate limiting, cheap “HTTP API” flavor	Budget Friendly

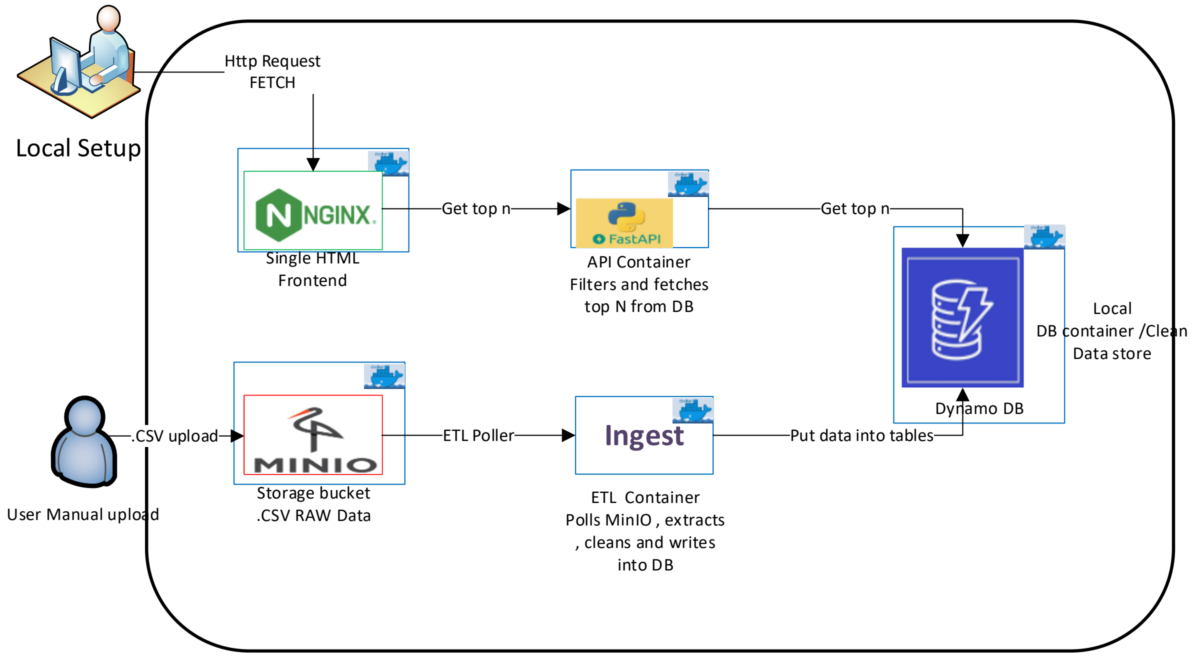
6	S3 Static Website + CloudFront	Host the UI	Fully serverless static hosting, global CDN if needed	Best Object Storage for Static page
7	IAM	Permissions between services	Principle of least privilege	Auto-created roles in Terraform
8	CloudWatch	Logs/metrics/alarms	Built-in visibility & alerting	Can Setup alarms for ingest failures, etc.

Assumptions, Constraints and Dependencies:

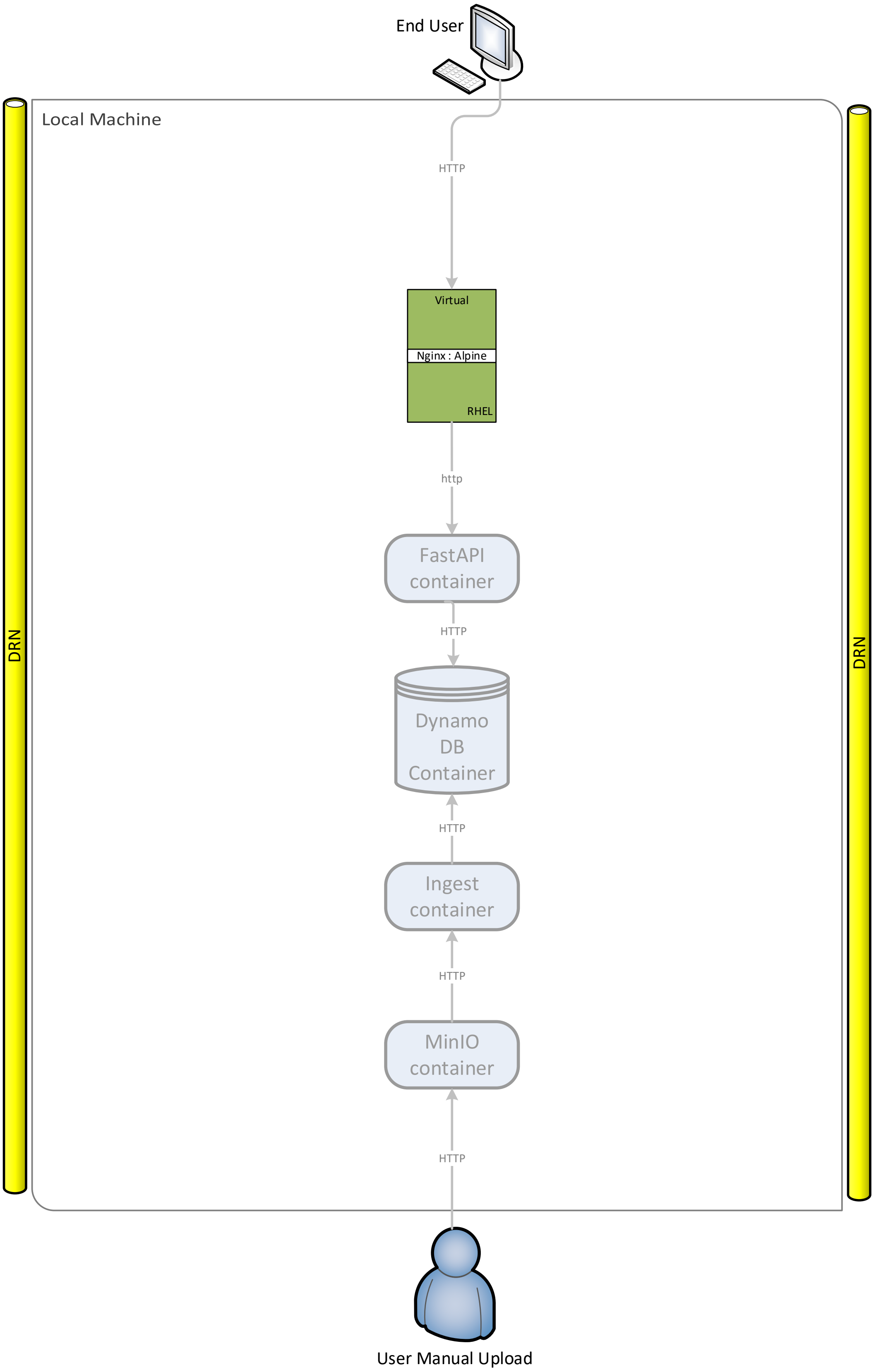
S.No	ACD	Description	Impact	Priority	Severity
1	Environment	No working environment, everything is done from scratch on the local machine. Tools and templates are not available	High	High	High
2	Excluded major factors	Since this is a POC kind of setup, major consideration like High availability, autoscaling, resilience, network, security (certificates, keys, tokens), performance, Operations, CICD, Observability, analytics & Reporting, backups, recovery, datalife cycle management, archives etc are excluded	High	High	High

3	Testing	The model is not tested with various test data samples. However , this showed results for small file size	High	High	High

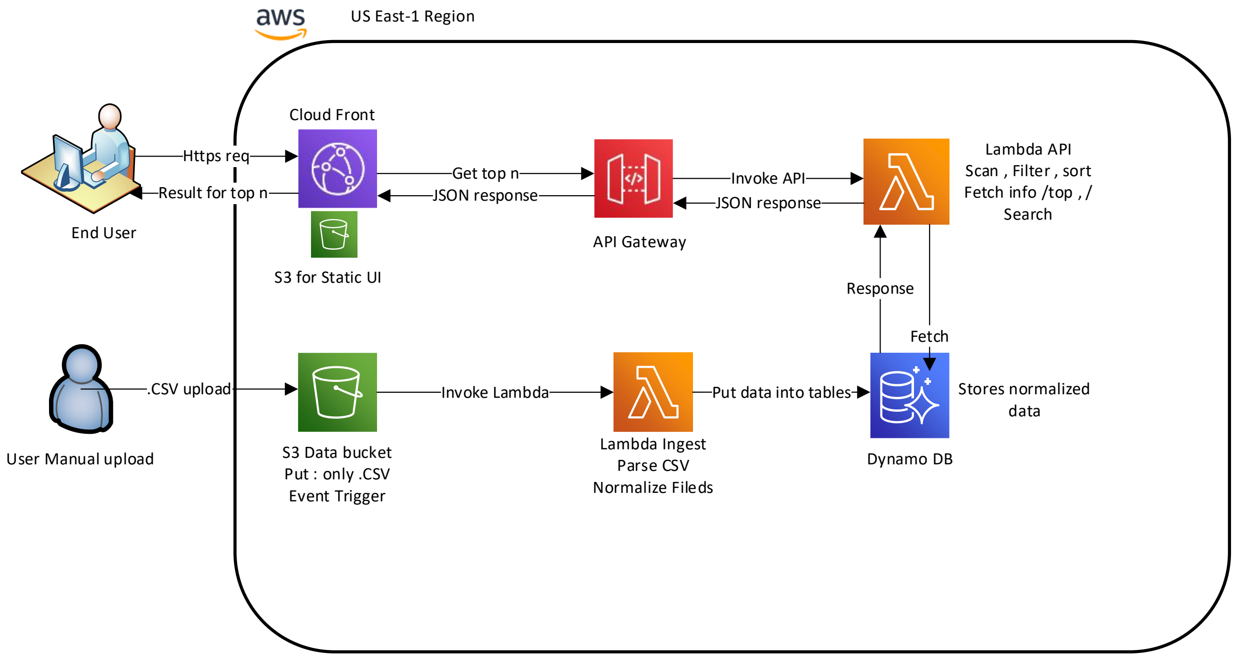
Architecture design for Local setup:



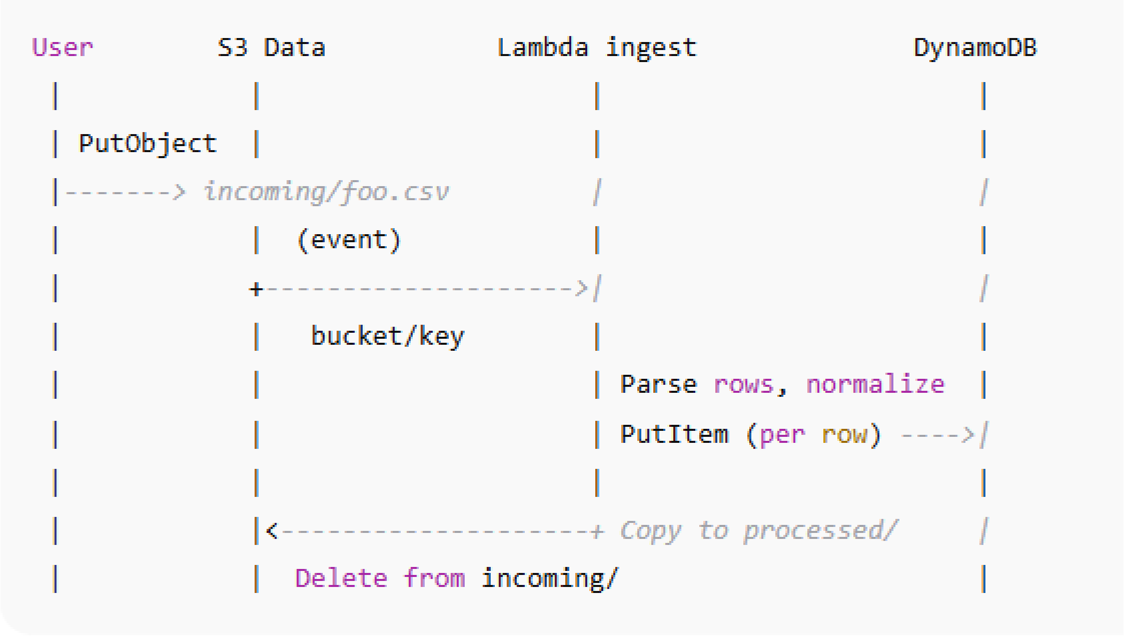
Infra view


End User
Local Machine
HTTP
Virtual
Nginx : Alpine
RHEL
http
FastAPI container
HTTP
Dynamo DB Container
HTTP
Ingest container
HTTP
MinIO container
HTTP
User Manual Upload
DRN
DRN

Architecture Design for AWS setup



Data Ingestion Sequence (CSV -> DB)

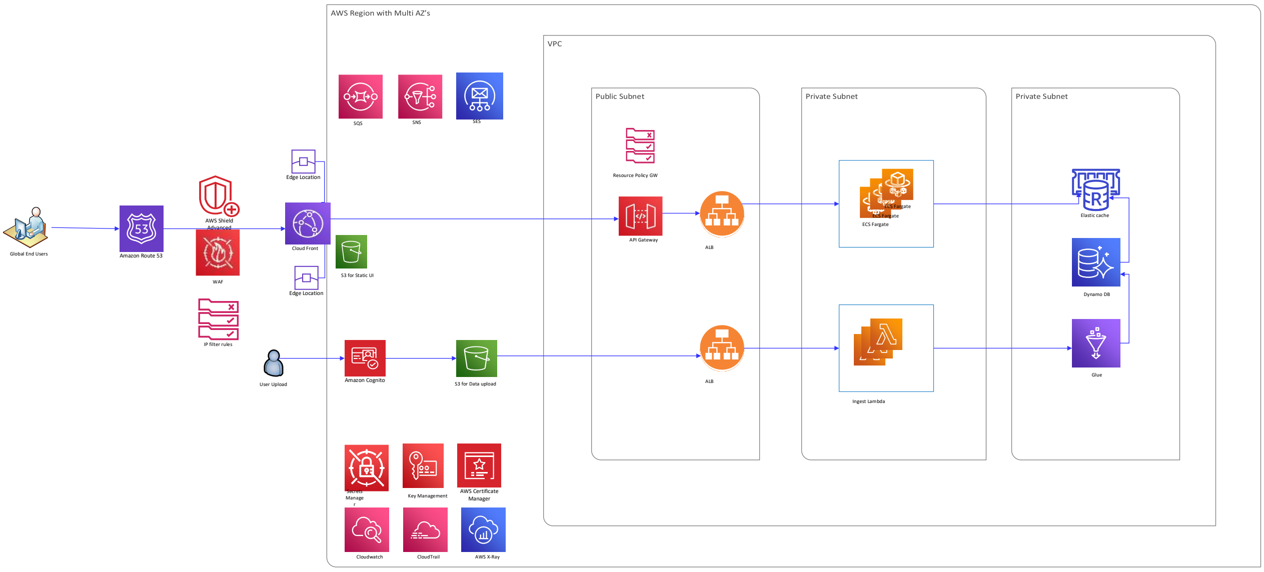


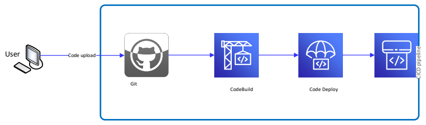
```
User          S3 Data            Lambda ingest              DynamoDB
 |             |                    |                          |
 | PutObject   |                    |                          |
 |-------> incoming/foo.csv         |                          |
 |             |  (event)           |                          |
 |             +------------------->|                          |
 |             |   bucket/key       |                          |
 |             |                    | Parse rows, normalize    |
 |             |                    | PutItem (per row) ---->  |
 |             |                    |                          |
 |             |<------------------+ Copy to processed/        |
 |             | Delete from incoming/                         |
```

Query Sequence (UI -> API -> DB)

```
Browser (CloudFront UI)        API Gateway          Lambda api           DynamoDB
         |                         |                    |                     |
GET /top?state=CA&limit=5          |                    |                     |
-------->                          |                    |                     |
         |------------------------>| (AWS_PROXY)        |                     |
         |                         |------------------->| Scan/Filter/Sort -->|
         |                         |<-------------------| Items               |
         |<------------------------| 200 JSON           |                     |
Show table                         |                    |                     |
```

As mentioned earlier, due to time, skills and resource constraints, a very simple design has been chosen for implementation. However, created a reference model in consideration of Network, security, availability, observability and deployment.


AWS Region with Multi AZ's
VPC
Public Subnet
Private Subnet
Private Subnet
Global End Users
Amazon Route 53
AWS Shield Advanced
WAF
IP filter rules
Edge Location
Cloud Front
Edge Location
S3 for Static UI
SQS
SNS
SES
User Upload
Amazon Cognito
S3 for Data upload
Secrets Manage r
Key Management
AWS Certificate Manager
Cloudwatch
CloudTrail
AWS X-Ray
Resource Policy GW
API Gateway
ALB
ALB
ECS Fargate
ECS Fargate
ECS Fargate
Ingest Lambda
Elastic cache
Dynamo DB
Glue


User
Code upload
Git
CodeBuild
Code Deploy
CICD pipeline