Key Principles to Include in the Architecture

| **Area** | **What to Include** | **Why It Satisfies the Customer** |
| --- | --- | --- |
| **Serverless First** | Use Lambda, Glue, Step Functions, AppFlow | No servers to manage or provision |
| **Auto-scaling & Fully Managed** | Use services like Glue, Athena, Redshift Serverless, EMR Serverless | Automatically handles load; no infra tuning |
| **No Infrastructure Code** | Enable Git-based deployment (CI/CD) with GitHub Actions or Jenkins | Devs push code, infra is provisioned automatically/or manual deployment |
| **Built-in Monitoring** | Use CloudWatch, X-Ray, or embed with Datadog/Splunk | No need to set up observability separately |
| **Zero ETL Option** | Use Redshift Zero-ETL with Aurora/Dynamo/S3 | No pipelines or job orchestration needed |
| **Pre-built Framework Templates** | Provide a plug-and-play code repo (e.g., with boilerplate PySpark, SQL, or Python Lambda code) | Lets devs start writing logic right away |
| **Data Contracts / Schema Registry** | Enforce schemas using Glue Schema Registry or AVRO/JSON validation | Keeps developers from worrying about integration issues |
| **Governance Handled** | Use Lake Formation & IAM Roles for access control | Security is abstracted away from developers |
| **Async & Event-Driven** | Use EventBridge or SQS for triggers | Devs don’t need to build cron or polling logic |
| **Developer Portal / Templates** | UI or CLI to scaffold new pipelines with Git templates | No time wasted on setup or standards |

**Core Framework Building Blocks (Abstracted Infra)**

| **Layer** | **Framework Component** |
| --- | --- |
| **Ingestion** | AppFlow / EventBridge Pipes / DMS (preconfigured templates) |
| **Processing** | Glue Job templates (SQL, PySpark), Lambda functions for micro-tasks |
| **Orchestration** | Step Functions with prebuilt state machines |
| **Storage** | S3 (raw, refined, curated folders auto-created) |
| **Metadata Management** | Glue Catalog + Tags, with templates |
| **Security/Governance** | Lake Formation (with preconfigured roles/policies) |
| **Deployment** | GitHub Actions / Jenkins that deploys on push |
| **Monitoring** | CloudWatch dashboards, alarms, embedded in template |
| **Cost Management** | Budgets + Usage reports pre-configured per project/account |

**End Result for the Customer**

| **Goal** | **How It’s Achieved** |
| --- | --- |
| **Zero Infra Worry** | All services are serverless/managed |
| **Developer-Only Focus** | Only code templates are exposed |
| **Secure and Governed** | Preconfigured Lake Formation, IAM roles |
| **Rapid Onboarding** | Templates and CLI scaffolders |
| **Scalable & Reliable** | Native AWS services with auto-retry, parallelism |
| **Extensible** | Add more sources/processors via plug-ins |

**code-only AWS data processing framework**.

Slide 1: Developer Experience & Entry Point

[Developer Portal / CLI]

|

v

[Select Source Type: API / DB / Event / File Upload]

|

v

[Select Processing Type: Lambda / Glue / SQL / ML]

|

v

[Auto-Generate Code Template (Git Repo)]

|

v

[Write Logic in transform.py OR sql\_logic.sql]

|

v

[Push to GitHub → CI/CD Auto Triggers or manual deployment]

|

v

[IAC / Job code is deployed]

Slide 2: Processing & Orchestration Layer (After Infra Is Auto-Created)

### **Purpose: Execute developer code-only logic with zero infra touch.**

[Step Function Start - "OrchestrateDataPipeline"]

|

v

[Step: Validate Input File]

- Lambda: validate\_input.py

- Checks file format, schema, expected columns, partition path

- If invalid → GoTo: [ErrorHandler]

|

v

[Choice: Route Based on Processor Type]

├── If "lambda":

│ └─> [Lambda Task - transform\_lambda.py]

│ - Executes user code

│ - Reads from S3/raw

│ - Writes to S3/refined

│ - Emits custom metrics

│

├── If "glue\_pyspark":

│ └─> [Glue Job - transform\_spark.py]

│ - Reads from S3/raw or Glue Catalog

│ - Applies business logic (PySpark)

│ - Writes to S3/refined (Parquet/Partitioned)

│ - Registers/updates Glue Catalog

│

├── If "glue\_sql":

│ └─> [Glue Job - sql\_logic.sql]

│ - Executes SQL-based logic

│ - Runs on DataBrew or Glue Studio engine

│

└── If "ml\_model":

└─> [Lambda Task - run\_inference.py]

- Loads model from S3 or SageMaker endpoint

- Applies predictions on batch data

- Writes back to S3

|

v

[Step: Validate Transformation Output]

- Compares input/output record count

- Checks null thresholds / partition correctness

- Updates metadata or emits CloudWatch metric

|

v

[Step: Move to Curated Zone]

- Lambda: move\_data.py

- Adds metadata tags, object lock, encrypts file

- Optional: triggers downstream consumers via EventBridge

|

v

[Succeed OR GoTo Alerting Step if Thresholds Missed]

### **Error Handler Path (If Any Step Fails)**

[Catch Block]

|

v

[Send to ErrorHandler Lambda]

- Logs error details to CloudWatch

- Writes error to S3/error/yyyy-mm-dd/jobname.json

- Sends SNS alert with error summary

- Optionally creates JIRA/GitHub Issue via webhook

|

v

[Mark Execution Status: FAILED]

### **Developer View (What They See & Control)**

| **Feature** | **Controlled by Dev** | **Infra-Backed** |
| --- | --- | --- |
| transform\_lambda.py | ✅ | 🔁 Lambda IAM & event source |
| transform\_spark.py | ✅ | 🔁 Glue job runtime settings |
| sql\_logic.sql | ✅ | 🔁 Auto-wired to tables/DBs |
| Step Function definition | ❌ (generated) | ✅ Named, versioned |
| Input/output S3 paths | ❌ (templated) | ✅ Enforced via pipeline config |
| Retry/backoff configs | ❌ | ✅ Set via SFN definition |
| Error handling flow | ❌ | ✅ Configured centrally |

Slide 3:**Code-Centric GitOps Pipeline (No Infra Touch)**

[Push Code to GitHub]

|

v

[GitHub Actions Triggered]

|

v

[Build → Validate Code → Lint → Unit Test]

|

v

[Deploy Code to Lambda/Glue/StepFunction]

|

v

[Trigger Step Function Execution]

|

v

[Pipeline Logs → CloudWatch / Slack / Splunk]

**Observability & Governance (Fully Managed)**

[CloudWatch Logs]

+--> Log insights for each job

+--> Dashboards: Success %, Failures, Run Time

[CloudWatch Metrics]

+--> Duration, Cost, Concurrency

[S3 Access Logs / Object Tags]

+--> Tracked via Lake Formation for audit

[Glue Catalog]

+--> Enforced schema, data classification, access control

[Alerts (Optional)]

+--> SNS → Email / Slack

+--> EventBridge Rules → Automated Actions

**Security & Access Control (Abstracted from Devs)**

[Lake Formation Access Policies]

+--> Data-level row/column access

+--> Project-based access control

[Glue Catalog Table Permissions]

+--> Separate dev, staging, prod permissions

[IAM Roles]

+--> Auto-created based on template

+--> No direct handling by developers

[S3 Bucket Policies]

+--> Block Public Access + KMS Encryption

[Audit Trail]

+--> CloudTrail + Lake Formation Logs + Athena Queries