Here is a detailed, **reusable AWS data processing framework**t:

* A consolidated architecture flow (reusable pattern)
* “Screenshot” explanations (what settings, why, and where they matter)
* How IaC/CI/CD ties it all together
* Focus on how every detail is reusable/templated for new data pipelines

## **🟦 Slide 1: Reusable Framework – The Big Picture**

**Reusable Framework Layers**:

┌─────────────────────────────────────────────────────────────┐

│ 1. Ingestion (S3, AppFlow, DMS, Events, File Drop) │

│ 2. Orchestration (Step Functions, event triggers) │

│ 3. Processing & DQ (Glue, Lambda, ML, Data Quality) │

│ 4. Storage & Catalog (S3 Tiers, Glue Catalog, Redshift) │

│ 5. Analytics & Delivery (Athena, QuickSight, Exports) │

│ 6. Observability (CloudWatch, Alarms, Dashboards) │

│ 7. Security & Audit (IAM, KMS, LF, CloudTrail, S3 Policy) │

│ 8. CI/CD & IaC (CodePipeline, CloudFormation, GitHub) │

│ 9. Cross-Platform (Lambda → GCP/Snowflake/On-prem) │

└─────────────────────────────────────────────────────────────┘

* **Reusable**: Each layer is modular, parameterized, and can be deployed for new pipelines via IaC templates.
* **Templated**: Parameterize source, logic, output, and security for each new workload.

## **🟦 Slide 2: Ingestion, Security, and Governance**

**1. S3 Ingestion Bucket (CFT Example & Screenshot Details):**

RawBucket:

Type: AWS::S3::Bucket

Properties:

BucketEncryption:

ServerSideEncryptionConfiguration:

- ServerSideEncryptionByDefault:

SSEAlgorithm: aws:kms

KMSMasterKeyID: !Ref MyKmsKey

VersioningConfiguration: { Status: Enabled }

LifecycleConfiguration:

Rules:

- Status: Enabled

Transitions:

- TransitionInDays: 30

StorageClass: GLACIER

**Screenshot Explanation:**

* **Encryption**: Enforced via aws:kms for compliance.
* **Versioning**: Protects against accidental deletion/overwrite.
* **Lifecycle Policy**: Controls cost by auto-archiving.

**2. S3 Bucket Policy – Secure Transport Only:**

RawBucketPolicy:

Type: AWS::S3::BucketPolicy

Properties:

Bucket: !Ref RawBucket

PolicyDocument:

Statement:

- Effect: Deny

Principal: "\*"

Action: s3:\*

Resource: !Sub "${RawBucket.Arn}/\*"

Condition: { Bool: { "aws:SecureTransport": "false" } }

**Screenshot Explanation:**

* **Prevents insecure (non-HTTPS) access**.
* Reusable for every S3 bucket via CFT parameter.

**3. Lake Formation Setup:**

LakeSettings:

Type: AWS::LakeFormation::DataLakeSettings

Properties:

CreateDatabaseDefaultPermissions:

- Principal: { DataLakePrincipalIdentifier: !Ref DataLakeAdmins }

Permissions: [ "ALL" ]

Admins: [ { DataLakePrincipalIdentifier: !Ref DataLakeAdmins } ]

**Screenshot Explanation:**

* **Lake Formation** enforces column/row-level security across all tables for all data lakes.

## **🟦 Slide 3: Orchestration, Processing, Data Quality**

**1. Step Functions (Reusable Orchestration, JSON Example):**

"States": {

"ValidateInput": { "Type": "Task", "Resource": "arn:aws:lambda:...:function:ValidateInput" },

"StartGlueJob": {

"Type": "Task",

"Resource": "arn:aws:states:::glue:startJobRun.sync",

"Next": "ValidateDQ"

},

"ValidateDQ": { "Type": "Task", "Resource": "arn:aws:lambda:...:function:DQCheck", "End": true }

}

**Screenshot Explanation:**

* Each state is parameterized; plug in new Lambda/Glue as pipeline evolves.
* **Reusable** for any ETL logic—just swap out the handler or Glue job name.

**2. Glue Job (CFT Example):**

TransformGlueJob:

Type: AWS::Glue::Job

Properties:

Command: { Name: glueetl, ScriptLocation: !Ref ScriptS3 }

GlueVersion: "4.0"

DefaultArguments:

"--TempDir": !Ref TempS3

"--enable-metrics": ""

"--job-language": "python"

MaxRetries: 2

**Screenshot Explanation:**

* **ScriptLocation** is a parameter; pipeline logic is drop-in.
* **Retries, logging, metrics** pre-configured for every job.

**3. Data Quality Lambda (Python):**

def handler(event, context):

# event: { row\_count, null\_percent, expected\_schema }

assert event["row\_count"] > 1000

assert event["null\_percent"] < 0.05

# More checks...

**Screenshot Explanation:**

* All checks are parameterized—DQ rules live in code/config for each pipeline.

## **🟦 Slide 4: Storage, Catalog, Analytics, Cross-Platform**

**1. Curated S3 & Glue Catalog Table (CFT):**

CuratedBucket:

Type: AWS::S3::Bucket

Properties:

BucketEncryption: ...

CuratedTable:

Type: AWS::Glue::Table

Properties:

TableInput:

Name: !Ref TableName

StorageDescriptor:

Location: !Sub "s3://${CuratedBucket}/curated/"

Columns: !Ref TableColumns

**Screenshot Explanation:**

* Each table/bucket is parameterized—add new datasets with new config only.

**2. Athena Query Example:**

SELECT \* FROM glue\_catalog.curated\_table WHERE dt='${date}'

**Screenshot Explanation:**

* Athena points to the cataloged, partitioned data, always up-to-date.

**3. Cross-Platform Lambda Example (GCP/Snowflake):**

import requests

def handler(event, context):

requests.post("https://gcp-func-url", json=event)

**Screenshot Explanation:**

* Lambda can invoke any external API, enabling cross-cloud orchestration.

## **🟦 Slide 5: Observability, Alerting, Automated Actions**

**1. CloudWatch Alarm (CFT):**

GlueJobFailureAlarm:

Type: AWS::CloudWatch::Alarm

Properties:

MetricName: FailedJobs

Namespace: AWS/Glue

Threshold: 1

AlarmActions: [!Ref AlarmTopic]

**Screenshot Explanation:**

* **Reusable**: Set up for every job/pipeline; Alarm actions parameterized.

**2. CloudWatch Dashboard (JSON):**

{

"widgets": [

{ "type": "metric", "properties": { "metrics": [["AWS/Glue", "SucceededJobs"]] } },

{ "type": "metric", "properties": { "metrics": [["AWS/Glue", "FailedJobs"]] } }

]

}

**Screenshot Explanation:**

* Dashboards built per-pipeline or consolidated at the account/org level.

**3. Error Remediation Lambda:**

def handler(event, ctx):

# On alarm, send to SNS/Jira/Slack

...

**Screenshot Explanation:**

* Hooks for automated ticketing, notification, or reprocessing.

## **🟦 Slide 6: Security, Audit, Cost Management**

**1. IAM Roles, Lake Formation Policies (CFT):**

GlueJobRole:

Type: AWS::IAM::Role

Properties:

ManagedPolicyArns:

- arn:aws:iam::aws:policy/service-role/AWSGlueServiceRole

Policies:

- PolicyName: LakeFormationAccess

PolicyDocument:

Statement:

- Effect: Allow

Action: lakeformation:GetDataAccess

Resource: "\*"

**Screenshot Explanation:**

* Role grants only specific access; **one template used across all jobs**.

**2. CloudTrail for Audit (CFT):**

AuditTrail:

Type: AWS::CloudTrail::Trail

Properties:

IsMultiRegionTrail: true

S3BucketName: !Ref AuditBucket

**Screenshot Explanation:**

* Full audit for all activities, **pre-integrated for all resources**.

**3. Cost Control – S3 Lifecycle, Glue Autoscaling (CFT):**

CuratedBucket:

Type: AWS::S3::Bucket

Properties:

LifecycleConfiguration:

Rules:

- Status: Enabled

Transition:

- TransitionInDays: 90

StorageClass: DEEP\_ARCHIVE

**Screenshot Explanation:**

* Every bucket can be configured with cost-saving policies by parameter.

## **🟦 Slide 7: CI/CD & IaC Automation (Reusable for Every Pipeline)**

**1. CodePipeline CFT:**

Pipeline:

Type: AWS::CodePipeline::Pipeline

Properties:

Stages:

- Name: Source

Actions: [{ ... }] # GitHub or CodeCommit source

- Name: Build

Actions: [{ ... }] # Lint/test/deploy code

- Name: Deploy

Actions: [{ ... }] # Deploy CloudFormation, update Lambda/Glue code

**Screenshot Explanation:**

* Triggered on every commit/merge; pipelines can be generated per workload.

**2. GitHub Actions Example:**

on: push

jobs:

deploy:

steps:

- uses: aws-actions/configure-aws-credentials@v1

- run: aws cloudformation deploy --template-file template.yaml --stack-name my-pipeline

**Screenshot Explanation:**

* Ensures code and infra always deploy together; totally reusable for new projects.

## **🟦 Slide 8: Extending & Reusing the Framework**

* All CFT templates take parameters: source, destination, data quality rules, compute sizing, retention, and security settings.
* New pipelines = new config file + one deploy command.
* **No manual setup**: code, logic, security, monitoring, and audit are all enforced at deploy time—**for every new data flow**.