

# Enhanced Design Document for Distributor Data Extraction Ingestion

Nathan Lunceford

2025-02-25

## Table of contents

<b>1</b>	<b>Overview</b>	<b>2</b>
<b>2</b>	<b>System Architecture</b>	<b>2</b>
2.1	Entry Point and Application Hosting . . . . .	2
2.2	Extraction Ingestion Orchestration . . . . .	3
2.3	External Dependencies . . . . .	3
<b>3</b>	<b>Design Patterns and Advanced Resilience Strategies</b>	<b>4</b>
3.1	Dependency Injection (DI) . . . . .	4
3.2	Registry Pattern . . . . .	4
3.3	Abstract Factory Pattern . . . . .	5
3.4	Builder Pattern . . . . .	5
3.5	Strategy Pattern . . . . .	5
3.6	Decorator Pattern . . . . .	6
3.7	Circuit Breaker Pattern . . . . .	6
3.8	Bulkhead Pattern . . . . .	7
3.9	Exponential Backoff with Jitter . . . . .	7
<b>4</b>	<b>Detailed Component Descriptions</b>	<b>7</b>
4.1	Program Class . . . . .	7
4.2	ERPService Class . . . . .	8
4.3	Providers . . . . .	9
4.4	Data Uploader – S3DataUploader . . . . .	9
4.5	Data Quality and Validation . . . . .	10
4.6	HashiCorp Vault Integration . . . . .	10
4.7	FerretDB Integration . . . . .	11
4.8	Feature Flag Management . . . . .	11

4.9	Metrics and Observability . . . . .	12
4.10	Data Lineage and Governance . . . . .	12
4.11	Configuration Models . . . . .	13
4.12	Development and Test Support . . . . .	13
<b>5</b>	<b>Pseudo-code for Nomad Integration</b>	<b>14</b>
<b>6</b>	<b>Advanced Error Handling and Recovery</b>	<b>23</b>
<b>7</b>	<b>Security and Compliance</b>	<b>23</b>
<b>8</b>	<b>Development and Operations Support</b>	<b>24</b>

## 1 Overview

This design document covers the extraction ingestion component of our ETL pipeline. In our architecture, distributors integrate with our system via two primary methods:

- **Direct Extraction Ingestion:** Where our system automatically provisions the correct resources based solely on the three supplied parameters: `--erp-type`, `--client-id`, and `--data-type`.
- **SFTP Ingestion:** Where distributors SFTP their data to our environment (this method is handled separately).

The design detailed here pertains only to the **direct extraction ingestion** method. This process extracts distributor data and deposits it into a pre-dropzone S3 bucket. At this stage, a minor transformation is applied: column names are standardized according to our global data dictionary and the output files are converted to Parquet format. More complex transformations occur later in the pipeline.

## 2 System Architecture

### 2.1 Entry Point and Application Hosting

- **Program Class:**
  - **Main Method:**
    - \* Parses command-line options (e.g., `--erp-type`, `--client-id`, `--data-type`) using `System.CommandLine`.
    - \* Sets up the dependency injection container and configures services.
    - \* Resolves the primary extraction ingestion service (`ERPService`) from the DI container and triggers the extraction process.

- **CreateHostBuilder Method:**
  - \* Configures the host with HashiCorp Vault, FerretDB services, core providers, registries, and builder components.
  - \* Configures **IHttpClientFactory** with resilience policies for API-based integrations.
  - \* Sets up feature flags, metrics collection, and health checks.

## 2.2 Extraction Ingestion Orchestration

- **ERPSERVICE Class:**
  - Acts as the orchestrator for the extraction ingestion process.
  - Retrieves credentials from HashiCorp Vault and configuration from FerretDB, respectively.
  - Dynamically resolves components for data extraction based on ERP type.
  - Uses **IHttpClientFactory** for managing HTTP connections in API mode.
  - Supports two extraction modes:
    - \* **API Mode:** For ERPs that expose APIs.
    - \* **Database Mode:** For ERPs that require direct database access.
  - Supports both batch processing and incremental extraction with change data capture (CDC).
  - Applies a minimal transform:
    - \* Standardizes column names according to our global data dictionary.
    - \* Applies data quality validations against predefined rules.
    - \* Converts all files to Parquet format with optional compression.
  - Deposits the extracted (and minimally transformed) data into a pre-dropzone S3 bucket.
  - Tracks data lineage for audit and compliance purposes.
  - (Note: Subsequent, more complex transformations are performed later in the overall ETL pipeline.)

## 2.3 External Dependencies

- **Infrastructure Services:**
  - **IVaultService:** Retrieves ERP credentials from HashiCorp Vault with least privilege access.
  - **IFerretDBService:** Fetches ERP configuration data from FerretDB.
  - **IAmazonS3:** Uploads the extracted data into the pre-dropzone S3 bucket.
  - **IFeatureFlagService:** Manages feature flags for gradual rollout of functionality.
  - **ICatalogService:** Integrates with data catalog for metadata management.

- **.NET Libraries:**
  - **System.CommandLine:** For command-line parsing.
  - **Microsoft.Extensions.Hosting & DI:** For hosting and dependency injection.
  - **System.Text.Json:** For JSON serialization/deserialization.
  - **IHttpClientFactory:** For managing HttpClient instances in API extraction mode.
  - **VaultSharp:** For interacting with HashiCorp Vault.
  - **MongoDB.Driver:** For FerretDB interactions (FerretDB uses MongoDB wire protocol).
  - **Polly:** For resilience patterns including circuit breakers and bulkheads.
  - **OpenTelemetry:** For distributed tracing and metrics.
  - **FluentValidation:** For data contract validation.

## 3 Design Patterns and Advanced Resilience Strategies

### 3.1 Dependency Injection (DI)

- **Usage:**
  - Decouples service construction from business logic.
  - Registers Vault client, FerretDB connection, providers, registries, builder components, and the main extraction service in the DI container.
  - Configures IHttpClientFactory and related services.
- **Benefits:**
  - Enhances testability and maintainability.
  - Promotes separation of concerns.

### 3.2 Registry Pattern

- **Components:**
  - ERPRegistry, ExtractorRegistry, TransformationRegistry, and UploaderRegistry.
- **Usage:**
  - Centralizes lookup for ERP-specific factories and strategies.
  - Dynamically resolves connectors, extractors, transformers, and uploaders based on ERP type or data type.
- **Benefits:**
  - Simplifies addition of new ERP integrations.
  - Reduces direct dependencies between the extraction process and concrete implementations.

### 3.3 Abstract Factory Pattern

- **Component:**
  - The `IERPFactory` interface (managed via `ERPRegistry`).
  - `IHttpClientFactory` for HTTP client management.
- **Usage:**
  - Encapsulates creation of ERP-specific components (e.g., connectors and jobs).
- **Benefits:**
  - Supports multiple ERP systems with varying implementations without altering extraction logic.

### 3.4 Builder Pattern

- **Components:**
  - **APIRequestBuilder & AuthenticationBuilder:**
    - \* Provide fluent interfaces to construct complex API request objects.
  - **DatabaseQueryBuilder:**
    - \* Dynamically constructs SQL queries for ERPs requiring direct database access.
  - **ExtractConfigBuilder:**
    - \* Builds extraction configurations supporting both full and incremental extracts.
- **Usage:**
  - The `DatabaseQueryBuilder` collects parameters and produces a `DatabaseQuery` object with a `GenerateSql` method.
  - The `APIRequestBuilder` works with `IHttpClientFactory` to construct properly configured HTTP requests.
  - The `ExtractConfigBuilder` creates configurations for different extraction modes.
- **Benefits:**
  - Enhances readability and modularity.
  - Supports multiple extraction modes seamlessly.

### 3.5 Strategy Pattern

- **Components:**
  - Interfaces such as `IExtractor`, `ITransformer`, and `IValidator`.

- **Usage:**
  - Encapsulate different implementations for data extraction, transformation, and validation.
  - Registries select the appropriate strategy at runtime.
- **Benefits:**
  - Provides flexibility to extend or change algorithms without impacting the overall system.

### 3.6 Decorator Pattern

- **Components:**
  - **VaultCredentialProviderDecorator:**
    - \* Adds caching behavior to Vault credential retrieval.
  - **MetricsDecorator:**
    - \* Wraps core services to collect performance metrics.
  - **EncryptionDecorator:**
    - \* Adds field-level encryption for sensitive data.
  - **DataQualityDecorator:**
    - \* Adds data quality checks to transformations.
- **Usage:**
  - Enhances existing components with additional functionality without modifying their core logic.
- **Benefits:**
  - Improves performance through caching.
  - Provides observability and enhances security.
  - Ensures data quality through validation.

### 3.7 Circuit Breaker Pattern

- **Components:**
  - **VaultCircuitBreaker, FerretDBCircuitBreaker, S3CircuitBreaker**
- **Usage:**
  - Prevents cascading failures by breaking the circuit when dependencies fail.
  - Automatically restores service when dependencies recover.

- **Benefits:**
  - Enhances system resilience during partial outages.
  - Prevents overwhelming failing services with requests.

### 3.8 Bulkhead Pattern

- **Components:**
  - Separate connection pools for different ERP types and operations.
  - Isolated resource pools for critical vs. non-critical operations.
- **Usage:**
  - Isolates failures to prevent system-wide degradation.
  - Allocates resources based on operation criticality.
- **Benefits:**
  - Prevents resource exhaustion.
  - Improves system stability during partial failures.

### 3.9 Exponential Backoff with Jitter

- **Usage:**
  - Implements intelligent retry strategies for all external service calls.
  - Adds randomization to prevent thundering herd problems.
- **Benefits:**
  - Prevents overwhelming recovering services.
  - Distributes retry attempts evenly over time.

## 4 Detailed Component Descriptions

### 4.1 Program Class

- **Main Method:**
  - Parses Nomad-supplied command-line arguments.
  - Builds the DI container via `CreateHostBuilder`.
  - Resolves and invokes `ERPService.ProcessERPData`.
  - Handles global errors for graceful failure.

- **CreateHostBuilder Method:**

- Configures services including Vault client, FerretDB connection, providers, registries, and builder components.
- Sets up resilience policies with circuit breakers and bulkheads.
- Configures feature flags, metrics collection, and health checks.
- Sets up OpenAPI documentation generation.

## 4.2 ERPService Class

- **Responsibilities:**

- Orchestrates the extraction ingestion process.
- Retrieves credentials and ERP configuration.
- Dynamically resolves ERP-specific components using registries.
- Chooses between API or Database extraction modes based on the ERP configuration.
- Supports both batch and incremental extraction modes.
- Applies a minor transformation with data quality validation.
- Deposits the transformed data into a pre-dropzone S3 bucket.
- Tracks data lineage for audit and compliance.

- **Key Method – ProcessERPData:**

- **Credential & Configuration Retrieval:**

- \* Uses `ICredentialProvider` to obtain short-lived, least-privilege credentials.
- \* Uses `IConfigurationProvider` to obtain ERP settings.

- **Dynamic Component Resolution:**

- \* Uses registries to resolve ERP-specific factories, extractors, transformers, and uploaders.

- **Integration Modes:**

- \* **API Mode:**

- Builds an API request via `APIRequestBuilder` with mutual TLS if supported.
- Extracts data via `IExtractor.Extract` with circuit breaker protection.

- \* **Database Mode:**

- Builds a SQL query using `DatabaseQueryBuilder`.
- Extracts data via `IExtractor.ExtractFromDatabase` with connection isolation.

- **Extraction Modes:**

- \* **Full Extract:** Retrieves all data for the specified type.
- \* **Incremental Extract:** Retrieves only changed data since the last extraction.



- **Self-Healing:**
  - \* Implements automatic recovery procedures for common failure scenarios.
- **Subsequent Steps:**
  - \* Applies the minor transform with data quality validation.
  - \* Records data lineage metadata.
  - \* Uploads the resulting data into a pre-dropzone S3 bucket via `IDataUploader`.

### 4.3 Providers

- **VaultCredentialProvider:**
  - Retrieves and deserializes credentials from HashiCorp Vault.
  - Implements caching with TTL-based invalidation.
  - Supports dynamic secret rotation with configurable TTL.
  - Generates least-privilege, operation-specific credentials.
- **FerretDBConfigProvider:**
  - Fetches ERP configuration from FerretDB and maps it to an `ERPConfiguration` object.
  - Uses MongoDB driver since FerretDB implements MongoDB wire protocol.
  - Supports read-preference strategies for replica sets.
  - Implements schema evolution for backward compatibility.

### 4.4 Data Uploader – `S3DataUploader`

- **Responsibilities:**
  - Formats and uploads the minimally transformed data into S3.
  - Applies data compression for efficient storage and transfer.
  - Converts files to Parquet format and ensures standardized column names.
  - Encrypts sensitive fields before upload.
  - Records data lineage metadata.
- **Key Methods:**
  - **Upload:** Manages the upload process with checksums and integrity verification.
  - **FormatData:** Applies the transformation with data quality checks.
  - **TrackLineage:** Records data provenance information.

## 4.5 Data Quality and Validation

- **DataContractValidator:**
  - **Responsibilities:**
    - \* Validates data against predefined schemas and rules.
    - \* Reports quality issues with detailed diagnostics.
    - \* Enforces data governance policies.
  - **Key Methods:**
    - \* **ValidateSchema:** Ensures data adheres to expected structure.
    - \* **ValidateValues:** Checks data values against business rules.
    - \* **GenerateReport:** Creates detailed validation reports.
- **DataMaskingService:**
  - **Responsibilities:**
    - \* Identifies and masks sensitive information (PII).
    - \* Supports various masking techniques (hashing, tokenization, etc.).
    - \* Maintains referential integrity across masked datasets.
  - **Key Methods:**
    - \* **IdentifySensitiveFields:** Automatically detects potential PII.
    - \* **ApplyMasking:** Applies appropriate masking techniques.
    - \* **VerifyMasking:** Ensures masking effectiveness.

## 4.6 HashiCorp Vault Integration

- **VaultService:**
  - **Responsibilities:**
    - \* Manages connection to HashiCorp Vault with circuit breaker protection.
    - \* Retrieves secrets with proper authentication.
    - \* Handles secret versioning and rotation.
    - \* Generates dynamic, least-privilege credentials.
  - **Key Methods:**
    - \* **GetSecret:** Retrieves a secret by path.
    - \* **GetDynamicSecret:** Retrieves a dynamic secret with lease management.
    - \* **RenewToken:** Handles token renewal to maintain access.
  - **Configuration:**
    - \* Support for multiple authentication methods (AppRole, Token, K8s).
    - \* Automatic token renewal.
    - \* Secret caching with TTL-based invalidation.
    - \* Mutual TLS for secure communication.

- **VaultConfiguration:**
  - Stores settings such as **Address**, **AuthMethod**, **RoleId**, **SecretId**, and authentication paths.
  - Includes retry and timeout settings with exponential backoff and jitter.

## 4.7 FerretDB Integration

- **FerretDBService:**
  - **Responsibilities:**
    - \* Manages connection to FerretDB using MongoDB driver.
    - \* Executes queries and maps results to domain models.
    - \* Handles connection pooling and resilience.
    - \* Supports schema evolution for backward compatibility.
  - **Key Methods:**
    - \* **GetConfiguration:** Retrieves configuration by ERP type and client ID.
    - \* **UpdateConfiguration:** Updates configuration documents.
    - \* **ExecuteQuery:** Executes a MongoDB query and returns the results.
    - \* **GetSchemaVersion:** Retrieves schema version information.
  - **Configuration:**
    - \* Support for replica sets and read preferences.
    - \* Connection pooling with bulkhead isolation.
    - \* Automatic retry with exponential backoff and jitter.
    - \* Circuit breaker protection against cascading failures.
- **FerretDBConfiguration:**
  - Stores settings such as **ConnectionString**, **Database**, **Collection**, and authentication credentials.
  - Includes timeout and retry settings.
  - Defines bulkhead configuration for connection isolation.

## 4.8 Feature Flag Management

- **FeatureFlagService:**
  - **Responsibilities:**
    - \* Manages feature flags for gradual rollout of new features.
    - \* Supports A/B testing and canary releases.
    - \* Provides runtime configuration without deployment.
  - **Key Methods:**

- \* **IsFeatureEnabled**: Checks if a feature is enabled for a specific context.
- \* **GetFeatureConfiguration**: Retrieves configuration for an enabled feature.
- \* **RecordFeatureUsage**: Tracks feature usage for analytics.
- **Configuration**:
  - \* Support for user, client, and global targeting.
  - \* Time-based and percentage-based rollouts.
  - \* Integration with monitoring for impact assessment.

## 4.9 Metrics and Observability

- **MetricsService**:
  - Collects performance metrics on key operations.
  - Integrates with Prometheus for metrics collection.
  - Supports custom dimensions for detailed analysis.
  - Records histogram metrics for latency distribution.
- **HealthCheckService**:
  - Provides health status of dependencies (Vault, FerretDB, S3).
  - Implements circuit breaker pattern for degraded services.
  - Exposes health endpoints for monitoring.
  - Supports self-healing procedures for common issues.
- **TracingService**:
  - Provides distributed tracing across system components.
  - Integrates with OpenTelemetry for standardized telemetry.
  - Correlates logs, metrics, and traces for holistic observability.
  - Supports sampling strategies for high-volume production environments.

## 4.10 Data Lineage and Governance

- **DataLineageService**:
  - **Responsibilities**:
    - \* Tracks data origin, transformations, and destinations.
    - \* Records metadata about extraction processes.
    - \* Supports audit requirements and compliance verification.
  - **Key Methods**:
    - \* **StartLineageRecord**: Creates a new lineage tracking record.
    - \* **RecordTransformation**: Logs applied transformations.
    - \* **CompleteLineageRecord**: Finalizes the lineage record.

- **Integration:**
  - \* Connects with data catalog for metadata management.
  - \* Provides lineage visualization through API.
  - \* Supports data governance and compliance reporting.

## 4.11 Configuration Models

- **ERPConfiguration:**
  - Stores settings such as `BaseUrl`, `CompanyId`, `WarehouseId`, `RequiredHeaders`, and `timeout/retry` settings.
  - **Fields for Database Access:**
    - \* `AccessType`: Enum (`API` or `Database`).
    - \* `ConnectionString`: For direct database connections.
    - \* `Schema`: Database schema.
    - \* `BatchSize`: Number of records to fetch per batch.
  - **Fields for Incremental Extract:**
    - \* `SupportsCDC`: Whether the ERP supports Change Data Capture.
    - \* `CDCConfiguration`: Settings for CDC-based extraction.
    - \* `WatermarkColumn`: Column used for incremental extraction.
- **ERPCredentials:**
  - Holds secure API keys and client secrets.
  - Includes operation-specific scoped credentials.
  - Supports dynamic credential generation.
- **UploadConfiguration:**
  - Used by the data uploader to configure the S3 upload.
  - Includes compression settings and encryption options.
  - Defines metadata for data catalog integration.
- **ResilienceConfiguration:**
  - Defines circuit breaker thresholds and recovery periods.
  - Configures retry policies with exponential backoff and jitter.
  - Specifies bulkhead isolation settings for resource pools.

## 4.12 Development and Test Support

- **LocalDevelopmentEnvironment:**
  - Provides containerized dependencies (Vault, FerretDB, S3-compatible storage).

- Supports mock ERP implementations for testing.
- Includes sample datasets for development.
- **IntegrationTestHarness:**
  - Facilitates automated testing against mock ERPs.
  - Supports scenario-based testing of the extraction process.
  - Includes performance benchmarking capabilities.

## 5 Pseudo-code for Nomad Integration

```

/// <summary>
/// Entry point for the ERP data extraction and ingestion process.
/// Processes command-line arguments from Nomad and orchestrates the ETL workflow.
/// </summary>
Main:
    // Parse Nomad-supplied command-line arguments:
    //   --erp-type, --client-id, --data-type
    options = parseArguments(["--erp-type", "--client-id", "--data-type"])

    // Build the host container with dependency injection configured
    host = createHostBuilder().build()

    // Register health checks for dependencies
    host.registerHealthChecks()
        .addCheck<VaultHealthCheck>("vault")
        .addCheck<FerretDBHealthCheck>("ferretdb")
        .addCheck<S3HealthCheck>("s3")

    // Setup OpenAPI documentation
    host.configureOpenApi(options => {
        options.Title = "Distributor Data Extraction API";
        options.Version = "v1";
        options.Description = "API for extracting distributor data from various ERP systems"
    })

    // Retrieve the ERPService from the DI container
    erpService = host.getService<ERPService>()

    // Trigger the extraction ingestion process with the supplied parameters
    erpService.ProcessERPData(options.erpType, options.clientId, options.dataType)

```

```

/// <summary>
/// Configures and builds the host with all necessary services.
/// </summary>
CreateHostBuilder:
    return HostBuilder()
        .ConfigureServices(services =>
        {
            // Configure Vault client with circuit breaker
            services.AddSingleton<IVaultClient>(provider =>
            {
                var vaultOptions = new VaultClientSettings(
                    "https://vault.example.com:8200",
                    new AppRoleAuthMethodInfo(roleId, secretId)
                );
                vaultOptions.RetrySettings = new RetrySettings {
                    Enabled = true,
                    MaxAttempts = 5,
                    BackoffType = BackoffType.ExponentialWithJitter
                };
                return new VaultClient(vaultOptions);
            });

            // Configure FerretDB connection with bulkhead isolation
            services.AddSingleton<IMongoClient>(provider =>
            {
                var settings = MongoClientSettings.FromConnectionString(
                    "mongodb://ferretdb.example.com:27017"
                );
                settings.RetryWrites = true;
                settings.RetryReads = true;
                settings.ServerSelectionTimeout = TimeSpan.FromSeconds(5);
                settings.MaxConnectionPoolSize = 100;
                return new MongoClient(settings);
            });

            // Configure feature flag service
            services.AddSingleton<IFeatureFlagService, FeatureFlagService>();

            // Register providers with caching decorators
            services.AddSingleton<ICredentialProvider, VaultCredentialProvider>();
            services.Decorate<ICredentialProvider, CachedCredentialProviderDecorator>();

```

```

services.AddSingleton<IConfigurationProvider, FerretDBConfigProvider>();

// Register data quality and validation services
services.AddSingleton<IDataContractValidator, DataContractValidator>();
services.AddSingleton<IDataMaskingService, DataMaskingService>();

// Register data lineage service
services.AddSingleton<IDataLineageService, DataLineageService>();

// Register registries
services.AddSingleton<IERPRegistry, ERPRegistry>();
services.AddSingleton<IExtractorRegistry, ExtractorRegistry>();
services.AddSingleton<ITransformationRegistry, TransformationRegistry>();
services.AddSingleton<IUploaderRegistry, UploaderRegistry>();

// Register builders
services.AddSingleton<IAPIRequestBuilder, APIRequestBuilder>();
services.AddSingleton<IDatabaseQueryBuilder, DatabaseQueryBuilder>();
services.AddSingleton<IAuthenticationBuilder, AuthenticationBuilder>();
services.AddSingleton<IExtractConfigBuilder, ExtractConfigBuilder>();

// Register S3 client for data upload
services.AddAWSService<IAmazonS3>();

// Register core service with decorators for cross-cutting concerns
services.AddSingleton<ERPService>();
services.Decorate<ERPService, MetricsERPServiceDecorator>();
services.Decorate<ERPService, DataQualityDecorator>();
services.Decorate<ERPService, EncryptionDecorator>();

// Configure HTTP clients with resilience policies using Polly
services.AddHttpClient("default")
    .AddTransientHttpErrorPolicy(builder =>
        builder.WaitAndRetryAsync(
            retryCount: 3,
            sleepDurationProvider: retryAttempt =>
                TimeSpan.FromSeconds(Math.Pow(2, retryAttempt)) +
                TimeSpan.FromMilliseconds(new Random().Next(0, 1000)), // Jitter
            onRetry: (outcome, timespan, retryAttempt, context) => {
                // Log retry attempt
                logger.LogWarning($"Retry {retryAttempt} for {context.PolicyKey}
            }

```



```

    ))
    .AddCircuitBreakerPolicy(builder =>
        builder.CircuitBreakerAsync(
            handledEventsAllowedBeforeBreaking: 5,
            durationOfBreak: TimeSpan.FromSeconds(30),
            onBreak: (outcome, breakDelay) => {
                logger.LogError($"Circuit broken for {breakDelay.TotalSeconds}s!");
            },
            onReset: () => {
                logger.LogInformation("Circuit reset!");
            }
        )
    );

    // Add OpenTelemetry tracing
    services.AddOpenTelemetryTracing(builder => {
        builder
            .SetResourceBuilder(ResourceBuilder.CreateDefault().AddService("erp-extractor"))
            .AddSource("erp-extractor")
            .AddHttpClientInstrumentation()
            .AddMongoDBInstrumentation()
            .AddAspNetCoreInstrumentation()
            .AddJaegerExporter();
    });
});

/// <summary>
/// Processes ERP data extraction and performs initial transformation.
/// </summary>
/// <param name="erpType">The type of ERP system to extract from</param>
/// <param name="clientId">The client identifier</param>
/// <param name="dataType">The type of data to extract</param>
/// <remarks>
/// This method handles both API and Database extraction modes. For API mode,
/// it constructs appropriate API requests with authentication. For Database mode,
/// it builds and executes SQL queries. In both cases, the extracted data is:
/// 1. Minimally transformed (column standardization)
/// 2. Validated against data contracts
/// 3. Converted to Parquet format with compression
/// 4. Uploaded to a pre-dropzone S3 bucket
/// </remarks>
ERPService.ProcessERPData(erpType, clientId, dataType):
    Log "Starting ETL extraction ingestion for client [clientId] using ERP [erpType]"

```

```

/// <summary>Start metrics collection for this operation</summary>
using (metricsTimer = MetricsService.StartTimer("erp_process_data",
                                                { "erp_type": erpType, "client_id": clientId }))
using (tracer = TracingService.StartTrace("ProcessERPData"))
{
    /// <summary>Start data lineage tracking</summary>
    lineage = DataLineageService.StartLineageRecord(erpType, clientId, dataType)

    /// <summary>Check feature flags for enabled features</summary>
    bool useIncrementalExtract = FeatureFlagService.IsFeatureEnabled("IncrementalExtract", clientId)
    bool useCompression = FeatureFlagService.IsFeatureEnabled("Compression", clientId)
    bool useFieldEncryption = FeatureFlagService.IsFeatureEnabled("FieldEncryption", clientId)

    /// <summary>Retrieve least-privilege credentials from HashiCorp Vault</summary>
    credentials = CredentialProvider.GetLeastPrivilegeCredentials(erpType, clientId, dataType)

    /// <summary>Retrieve configuration from FerretDB</summary>
    erpConfig = ConfigurationProvider.GetConfiguration(erpType, clientId)

    /// <summary>
    /// Lookup common components via registries:
    /// - ERP-specific factory (for connectors and jobs)
    /// - Data extractor (for API or DB extraction)
    /// - Data transformer (to standardize columns and convert to Parquet)
    /// - Data validator (to validate data quality)
    /// - Data uploader (to upload data to the pre-dropzone S3 bucket)
    /// </summary>
    factory = ERPRegistry.GetFactory(erpType, clientId)
    extractor = ExtractorRegistry.GetExtractor(erpType)
    transformer = TransformationRegistry.GetStrategy(erpType, dataType)
    validator = ValidatorRegistry.GetValidator(erpType, dataType)
    uploader = UploaderRegistry.GetUploader("s3")

    /// <summary>Build extraction configuration based on mode</summary>
    extractConfig = ExtractConfigBuilder.New()
        .ForERP(erpType)
        .ForClient(clientId)
        .ForDataType(dataType)
        .UseIncrementalExtract(useIncrementalExtract && erpConfig.SupportsCDC)
        .WithLastExtractTime(useIncrementalExtract ? GetLastExtractTime(erpType, clientId) : null)
        .WithBatchSize(erpConfig.BatchSize)
        .Build()
}

```

```

/// <summary>
/// Handle database extraction mode
/// Builds and executes SQL queries for direct database access
/// </summary>
if erpConfig.AccessType == Database then:
    queryBuilder = DatabaseQueryBuilder()
        .ForERP(erpType)
        .WithConnectionString(erpConfig.ConnectionString)
        .WithSchema(erpConfig.Schema)
        .WithTable(dataType + "_table")
        .WithColumns("id", "created_at", "data")

    if useIncrementalExtract && erpConfig.SupportsCDC:
        queryBuilder.WithWhere(erpConfig.WatermarkColumn, ">", extractConfig.LastExt
    else:
        queryBuilder.WithWhere("is_processed", false)

    query = queryBuilder
        .WithOrderBy("created_at")
        .WithLimit(erpConfig.BatchSize)
        .WithCommandTimeout(erpConfig.TimeoutSeconds)
        .Build()

    Log "Executing database query: " + query.GenerateSql()

    // Use bulkhead isolation for database connection
    using (bulkhead = BulkheadPolicy.Execute(erpType + "-database", () => {
        extractedData = extractor.ExtractFromDatabase(query, extractConfig)
        return extractedData
    }))

/// <summary>
/// Handle API extraction mode
/// Constructs and executes authenticated API requests with resilience
/// </summary>
else:
    authBuilder = AuthenticationBuilder()
        .WithApiKey(credentials.ApiKey)
        .WithClientId(credentials.ClientId)
        .WithClientSecret(credentials.ClientSecret)

    if erpConfig.SupportsMutualTLS:

```

```

        authBuilder.WithClientCertificate(credentials.ClientCertificate)

auth = authBuilder.Build()

requestBuilder = APIRequestBuilder()
    .ForERP(erpType)
    .WithEndpoint(erpConfig.BaseUrl + "/api/v2/sales")
    .WithMethod(GET)
    .WithAuthentication(auth)
    .WithHeaders(erpConfig.RequiredHeaders)

if useIncrementalExtract && erpConfig.SupportsCDC:
    requestBuilder.WithQueryParameters({
        "companyId": erpConfig.CompanyId,
        "warehouse": erpConfig.WarehouseId,
        "pageSize": erpConfig.PageSize.ToString(),
        "changedSince": extractConfig.LastExtractTime.toISOString()
    })
else:
    requestBuilder.WithQueryParameters({
        "companyId": erpConfig.CompanyId,
        "warehouse": erpConfig.WarehouseId,
        "pageSize": erpConfig.PageSize.ToString()
    })

request = requestBuilder
    .WithRetryPolicy(erpConfig.MaxRetries)
    .WithTimeout(erpConfig.TimeoutSeconds)
    .Build()

Log "Executing API request to " + erpConfig.BaseUrl + "/api/v2/sales"

// Execute with circuit breaker protection
extractedData = CircuitBreakerPolicy
    .ForService("erp-api-" + erpType)
    .Execute(() => extractor.Extract(request, extractConfig))

/// <summary>Validate data against contract</summary>
Log "Validating extracted data against contract"
validationResult = validator.Validate(extractedData)

if !validationResult.IsValid:

```

```

        // Handle data quality issues based on severity
        if validationResult.HasCriticalIssues():
            throw new DataContractException(
                "Critical data quality issues detected: " +
                validationResult.GetCriticalIssuesSummary()
            )
        else:
            // Log warnings but continue processing
            Log "Warning: Data quality issues detected: " + validationResult.GetIssuesSummary()

    /// <summary>Transform the extracted data with data lineage tracking</summary>
    Log "Starting data transformation"
    transformedData = transformer.Transform(extractedData)
    lineage.RecordTransformation("ColumnStandardization", "Standardized columns according to schema")

    /// <summary>Apply field encryption for sensitive data if enabled</summary>
    if useFieldEncryption:
        transformedData = EncryptionService.EncryptSensitiveFields(
            transformedData,
            GetSensitiveFieldsConfig(erpType, dataType)
        )
        lineage.RecordTransformation("FieldEncryption", "Encrypted sensitive fields")

    /// <summary>Apply masking for non-production environments</summary>
    if environmentType != Production:
        transformedData = DataMaskingService.ApplyMasking(
            transformedData,
            GetDataMaskingConfig(erpType, dataType)
        )
        lineage.RecordTransformation("DataMasking", "Applied data masking for non-production")

    /// <summary>
    /// Configure and execute the S3 upload operation
    /// Data is stored in a pre-dropzone bucket with standardized path structure
    /// </summary>
    currentTimestamp = getCurrentTimestamp()

    uploadConfig = new UploadConfiguration(
        Bucket: "erp-data-" + clientId,
        Key: erpType + "/" + dataType + "/" + currentTimestamp + "/data.parquet",
        Format: Parquet,
        Compress: useCompression,
    )

```

```

        CompressionType: useCompression ? "SNAPPY" : null,
        Metadata: {
            "erp_type": erpType,
            "client_id": clientId,
            "data_type": dataType,
            "extract_timestamp": currentTimestamp,
            "lineage_id": lineage.Id,
            "extract_mode": useIncrementalExtract ? "incremental" : "full",
            "record_count": transformedData.Count,
            "schema_version": "1.2"
        }
    )

    Log "Starting data upload to S3 pre-dropzone"
    uploadResult = uploader.Upload(transformedData, uploadConfig)

    /// <summary>Update data catalog with metadata</summary>
    CatalogService.UpdateDatasetMetadata(
        datasetId: erpType + "-" + clientId + "-" + dataType,
        metadata: {
            "lastUpdated": currentTimestamp,
            "recordCount": transformedData.Count,
            "fileLocation": uploadResult.Location,
            "fileSize": uploadResult.Size,
            "schemaVersion": "1.2",
            "lineageId": lineage.Id
        }
    )

    /// <summary>Complete lineage record</summary>
    lineage.SetDestination(uploadResult.Location)
    lineage.CompleteLineageRecord()

    /// <summary>Update last extract time for incremental extracts</summary>
    if useIncrementalExtract:
        StoreLastExtractTime(erpType, clientId, dataType, currentTimestamp)

    Log "Extraction ingestion process completed successfully"
}

```

## 6 Advanced Error Handling and Recovery

- **Error Classification:**
  - Categorizes errors as transient or persistent.
  - Applies different recovery strategies based on error type.
  - Records error patterns for proactive monitoring.
- **Self-Healing Procedures:**
  - **Common Issue Resolution:**
    - \* Automatic token renewal for expired credentials.
    - \* Connection pool refresh for stale connections.
    - \* Temporary file cleanup for storage-related issues.
  - **Degraded Mode Operation:**
    - \* Falls back to full extract if incremental extract fails.
    - \* Disables optional features during high load.
    - \* Implements progressive backoff for system recovery.
- **Comprehensive Logging:**
  - Structured logging with correlation IDs across components.
  - Context-enriched log entries for easier debugging.
  - Log level adjustment based on operation criticality.
  - Integration with log aggregation systems.

## 7 Security and Compliance

- **Least Privilege Access:**
  - Dynamic generation of operation-specific credentials.
  - Short-lived tokens with minimal required permissions.
  - Credential scoping based on operation context.
  - Role rotation and separation of duties.
- **Data Protection:**
  - Field-level encryption for sensitive data.
  - Data masking for non-production environments.
  - Secure credential management with HashiCorp Vault.
  - Mutual TLS for secure service communication.
- **Audit and Compliance:**
  - Comprehensive data lineage tracking.

- Access and operation audit logging.
- Compliance validation against regulatory requirements.
- Regular security posture assessment.

## 8 Development and Operations Support

- **Local Development Environment:**

- Docker Compose setup with all dependencies.
- Mock ERPs for integration testing.
- Configuration templates for different scenarios.
- Development tools for data visualization and debugging.

- **Deployment and Infrastructure:**

- GitOps-based deployment pipelines.
- Infrastructure as Code for all components.
- Canary deployment support for risk mitigation.
- Blue-green deployment capability for zero downtime.

- **Documentation and Knowledge Sharing:**

- OpenAPI documentation for all service interfaces.
- Architecture decision records (ADRs) for design choices.
- Runbooks for common operational procedures.
- Automated documentation generation from code.