Proposed Solution

Task 1: Document Classification System - Use Case

Context

A bank seeks to automate the classification of customer-submitted PDF documents, like statements, contracts, and applications, by detecting their type and storing them correctly.

System Requirements

Input (1.1)

Documents arrive via:

- Email.
- Web portal.
- Mobile app.
- Third-party API.

Process (1.2)

- **Extract content**: Read text and analyze images.
- **Classify**: Use AI to identify document type.
- **Store**: Save in the right repository with metadata.

Output (1.3)

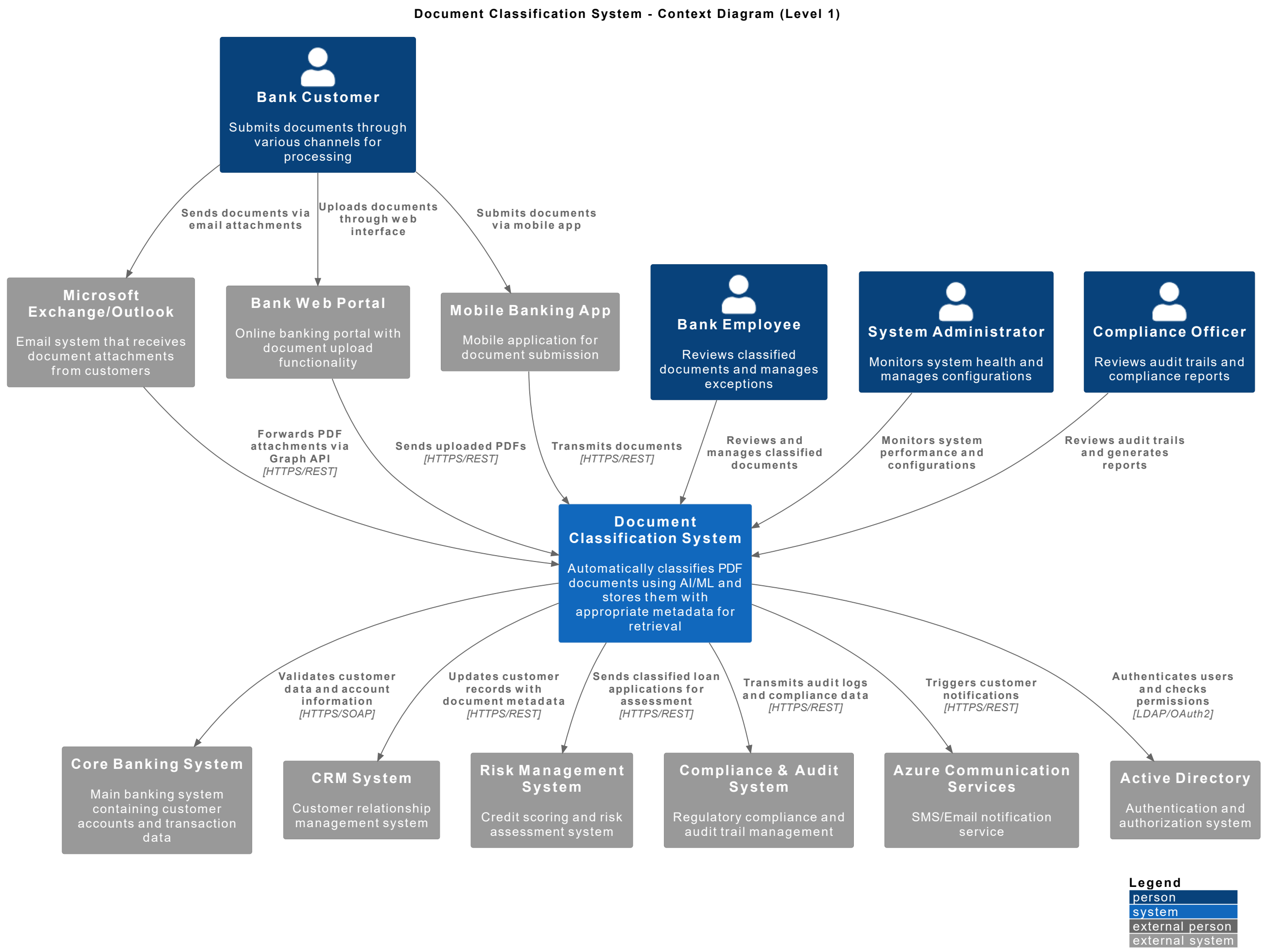
- Tagged by type (e.g., Contract, Statement)
- Indexed for search
- Logged for compliance

C4 Model Architecture

This solution uses the **C4 Model** with multiple layers to explain all system interactions:

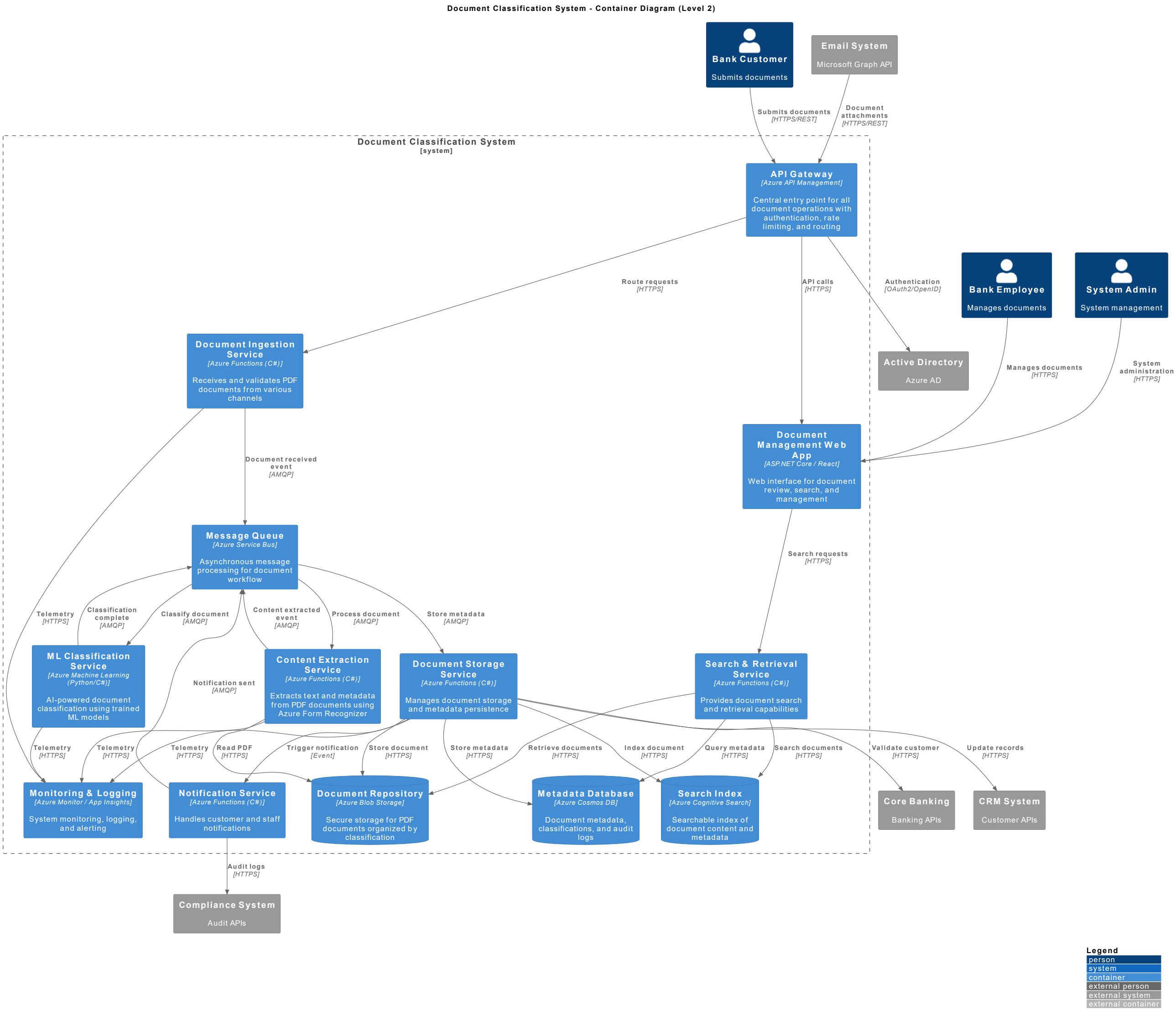
1. Context Diagram (Level 1)

Shows the system in its environment with external actors and systems.


Document Classification System - Context Diagram (Level 1)
Bank Customer
Submits documents through various channels for processing
Sends documents via email attachments
Uploads documents through web interface
Submits documents via mobile app
Microsoft Exchange/Outlook
Email system that receives document attachments from customers
Bank Web Portal
Online banking portal with document upload functionality
Mobile Banking App
Mobile application for document submission
Bank Employee
Reviews classified documents and manages exceptions
System Administrator
Monitors system health and manages configurations
Compliance Officer
Reviews audit trails and compliance reports
Forwards PDF attachments via Graph API
[HTTPS/REST]
Sends uploaded PDFs
[HTTPS/REST]
Transmits documents
[HTTPS/REST]
Reviews and manages classified documents
Monitors system performance and configurations
Reviews audit trails and generates reports
Document Classification System
Automatically classifies PDF documents using AI/ML and stores them with appropriate metadata for retrieval
Validates customer data and account information
[HTTPS/SOAP]
Updates customer records with document metadata
[HTTPS/REST]
Sends classified loan applications for assessment
[HTTPS/REST]
Transmits audit logs and compliance data
[HTTPS/REST]
Triggers customer notifications
[HTTPS/REST]
Authenticates users and checks permissions
[LDAP/OAuth2]
Core Banking System
Main banking system containing customer accounts and transaction data
CRM System
Customer relationship management system
Risk Management System
Credit scoring and risk assessment system
Compliance & Audit System
Regulatory compliance and audit trail management
Azure Communication Services
SMS/Email notification service
Active Directory
Authentication and authorization system
Legend
person
system
external person
external system

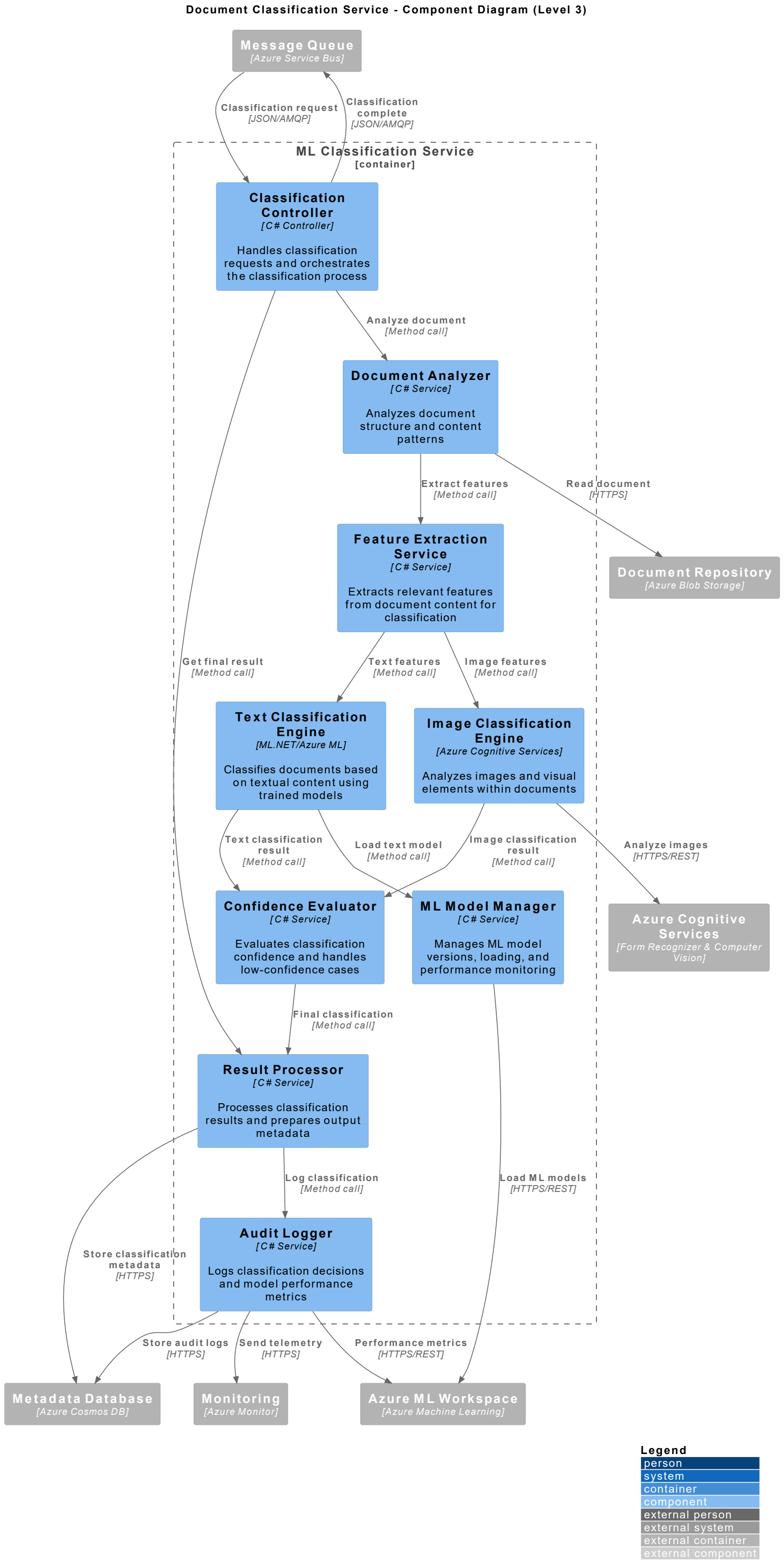
2. Container Diagram (Level 2)

Shows high-level technical building blocks and their interactions.



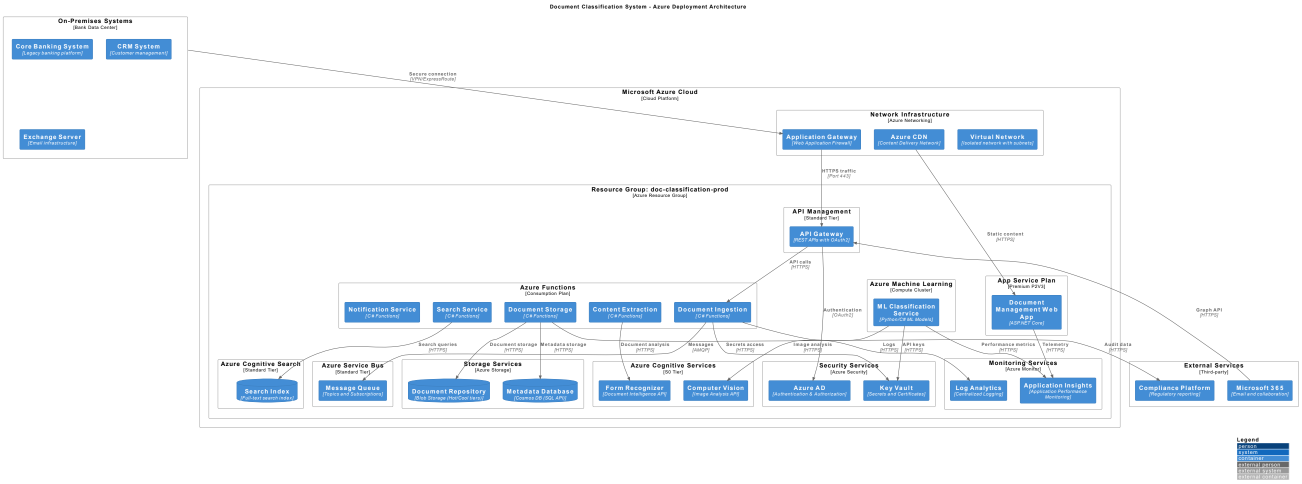
3. Component Diagram (Level 3)

Shows internal components within key containers.



4. Deployment Diagram

Shows infrastructure and deployment architecture on Microsoft Azure.



Key Components & Integrations

Core System Components

The solution is built from modular services that work together to classify and store documents efficiently:

- **API Gateway**: Entry point for document submissions.
- **Ingestion Service**: Receives and pre-processes PDFs.
- **ML Engine**: Uses AI to classify documents.
- **Document Repository**: Secure archive for storing files.
- **Metadata Manager**: Handles document tags and indexing.
- **Search Service**: Enables quick and accurate retrieval.

External Integrations

To connect with key systems and ensure smooth operations, the platform integrates with:

- **Email API**: Microsoft Graph for processing attachments.
- **Core Banking API**: Syncs with internal bank systems.
- **CRM API**: Connects customer data for context.
- **Compliance API**: Supports audits and reporting.
- **Notifications**: Azure Communication for user alerts.
- **Cognitive Services**: Powers AI document analysis.

Azure Technology Stack

The solution is cloud-native, built on Microsoft Azure components:

- API gateway via **Azure API Management**.
- Scalable compute with **Azure Functions**.
- AI-powered insights from **Cognitive Services**.
- Custom ML models with **Azure Machine Learning**.
- Secure storage in **Azure Blob Storage**.
- Metadata housed in **Azure Cosmos DB**.
- Search powered by **Azure Search**.
- Messaging via **Azure Service Bus**.
- Secrets managed with **Azure Key Vault**.
- Real-time monitoring using **Azure Monitor**.

Task 2: FinHub Data Processing Solution - From HLD to LLD

Overview

FinHub needs a data platform to collect financial data from multiple APIs (BCRA, ARCA, etc.), run ETL workflows, and deliver analytics to business leaders.

2.1 High-Level Design (HLD)

Architecture Overview

1. Ingestion

- Central API Gateway.
- Connectors for BCRA, ARCA, market data.
- Scheduling service and data validation.

2. Processing

- ETL Orchestrator.
- Data cleaning and enrichment.
- Business rule application.
- Quality checks and alerts.

3. Storage

- Data Lake for raw info.
- Data Warehouse for processed results.
- Metadata and archive layers.

4. Analytics & Reports

- Analytics Engine.
- Scheduled & on-demand reports.
- Live dashboards.
- Exposed via REST APIs.

5. Security & Compliance

- User authentication.
- Audit logs.
- Encryption for data at rest and in transit.
- GDPR/SOX compliance monitoring.

6. Operations

- Monitoring and alerts.
- Central config.

- Backup and recovery.

Data Sources & Flow

Sources

- Central banks, tax agencies, market feeds.
- Internal systems and credit bureaus.

Flow

Ingest → Validate → Transform → Store → Analyze → Report.

Design Considerations

Scalability

- Microservices and auto-scaling.
- Redis caching and load balancing.

Security

- Zero Trust model.
- API protection via OAuth2.
- Secure networking and secret storage.

Compliance

- Governance, retention policies.
- Full audit trails.
- GDPR data privacy tools.

2.2 Low-Level Design (LLD)

Selected Components - Detailed View

Data Ingestion Service

Designed with modularity and flexibility in mind, this service uses factory and strategy patterns to handle multiple data sources.

Main Classes

- `DataIngestionController` : Receives API requests.
- `DataOrchestrator` : Oversees the ingestion flow.
- `DataConnectorFactory` : Instantiates connectors based on source type.
- Specialized connectors: `BCRAConnector` , `ARCAConnector` , `MarketDataConnector` .
- `DataValidator` : Checks data integrity and applies rules.
- `AzureStorageService` : Manages cloud storage.
- `ServiceBusMessageQueue` : Publishes events asynchronously.

Design Patterns Used

- Factory: For connector creation.
- Strategy: Custom validation per source.
- Dependency Injection: Improves modularity and testability.
- Repository: Abstracts data access.

Class Interactions

- Controller depends on `Orchestrator` and `Validator` .
- Orchestrator manages flow using `Factory` , `Validator` , `Storage` , `Queue`
- Connectors implement a shared `IDataConnector` interface.

Data Processing Engine

A robust ETL pipeline, built to support transformation, rule application, and quality control.

Main Components

- `ETLOrchestrator` : Manages full ETL lifecycle.
- `TransformationService` : Performs data cleanup and enhancement.
- `TransformationEngine` : Executes transformation rules
- `BusinessRulesEngine` : Applies financial logic.
- `FinancialCalculationService` : Handles calculations.
- `DataQualityMonitor` : Flags anomalies.
- `SynapseDataWarehouseService` : Stores results.
- `ProcessingJobManager` : Manages ETL jobs.

Patterns Applied

- Chain of Responsibility: Rule processing flow.
- Strategy: Dynamic rule categories.
- Observer: Real-time quality alerts.
- Command: Task execution and tracking.
- Template Method: Structured ETL steps.

Workflow Summary

1. Ingestion request hits `ETLOrchestrator`.
2. Data gets validated.
3. Transformation and enrichment are applied.
4. Business rules and calculations executed.
5. Quality checks run.
6. Processed data stored with full lineage.

2.3 Architectural Enabler Features

1. Azure Data Lake Storage Gen2 Integration

Purpose
To handle large volumes of financial data with scalable, secure storage optimized for analytics and regulatory compliance.

Design Highlights

- **Hierarchical Namespace**: Faster directory-level operations for analytics.
- **Multi-Protocol**: Compatible with Blob APIs and HDFS.
- **Tiered Storage**: Lifecycle automation across Hot, Cool, Archive.
- **Security**: Azure AD integration, encryption, private endpoints.
- **Performance**: Supports parallel access and large file handling.

Implementation Phases

1. Partitioned setup.
2. Lifecycle and tiering automation.
3. Security and compliance hardening.
4. Performance tuning and monitoring.

Key Features

- Auto-partitioning by date/source.
- Smart tiering based on access patterns.
- Synapse integration for analytics.
- Audit logs and granular access control.

2. Azure Service Bus Message Queue

Purpose
To ensure reliable, ordered delivery of financial messages across decoupled services.

Design Highlights

- **Topics/Subs**: Multi-consumer message routing.
- **Message Sessions**: Maintain transaction order.
- **Dead Letter Queues**: Retry failed messages.
- **Duplicate Detection**: Ensure data consistency.
- **Auto-Scaling**: Adjusts to queue load dynamically.

Implementation Phases

1. Core messaging setup.
2. Advanced features (DLQ, sessions, deduplication).
3. Monitoring & alerting.

4. Performance and scaling optimization.

Key Features

- At-least-once delivery.
- Retry logic with exponential backoff.
- Correlated messaging for data tracing.
- Full observability via Azure Monitor.

3. Monitoring with Azure Monitor

Purpose
To ensure visibility into system health, performance, and data quality, with alerts and automation for proactive issue resolution.

Design Highlights

- **Multi-Layer Monitoring**: Infra, app, business metrics.
- **Real-Time Alerts**: Issues surfaced immediately.
- **Distributed Tracing**: Full visibility across services.
- **Custom Dashboards**: KPIs tailored to business needs.
- **Automation**: Enables self-healing workflows.

Implementation Phases

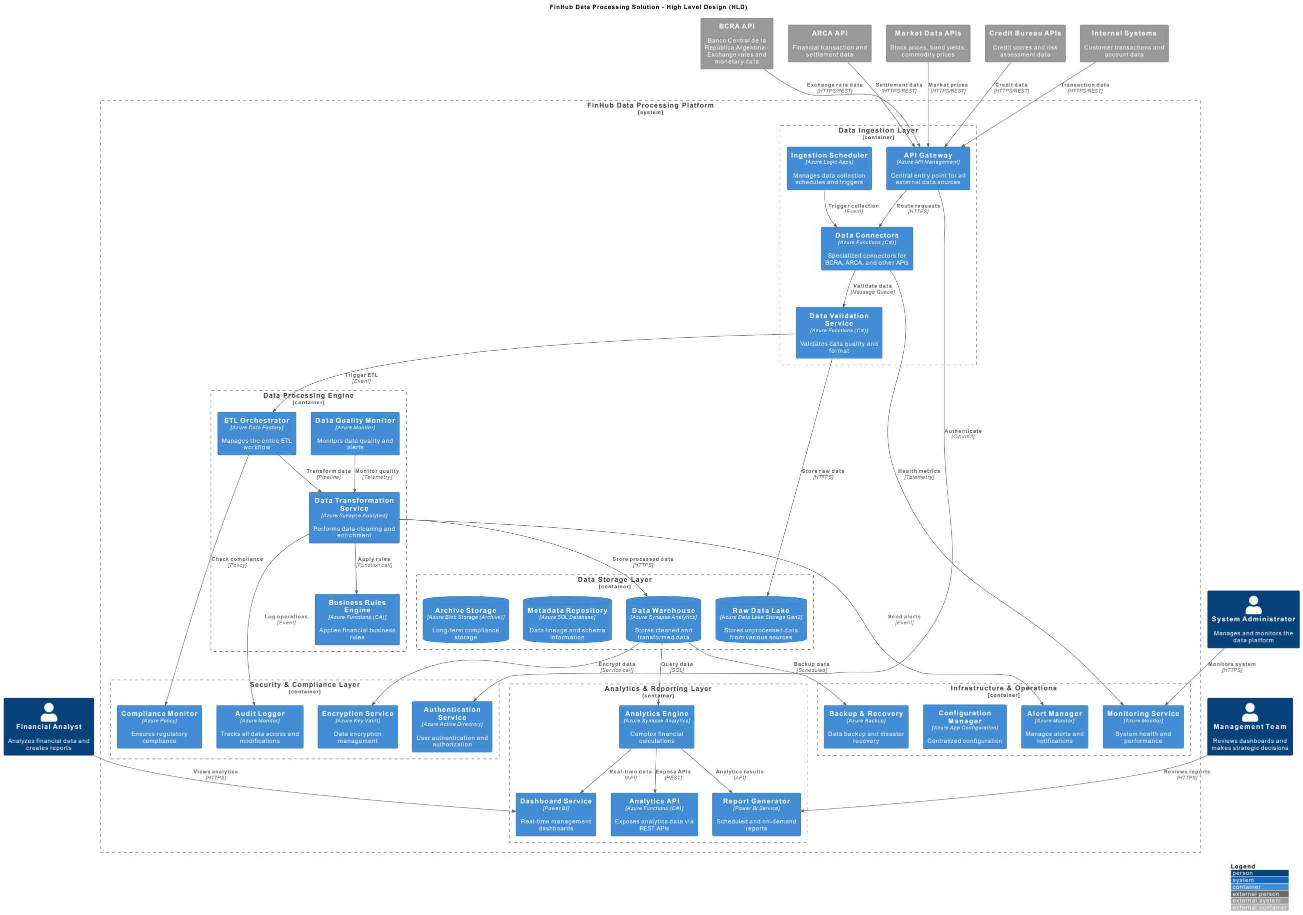
1. Basic system monitoring.
2. Business metrics and quality tracking.
3. Predictive alerts and analytics.
4. Automated remediation.

Key Features

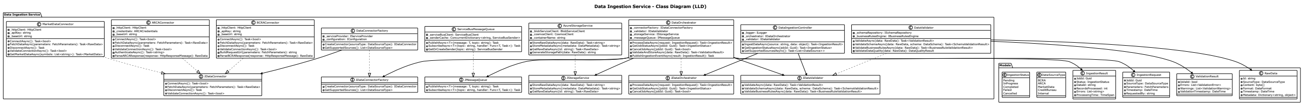
- Telemetry for data pipelines.
- Anomaly detection on quality signals.
- KPI dashboards for executives.
- Integration with incident management tools.

Diagrams

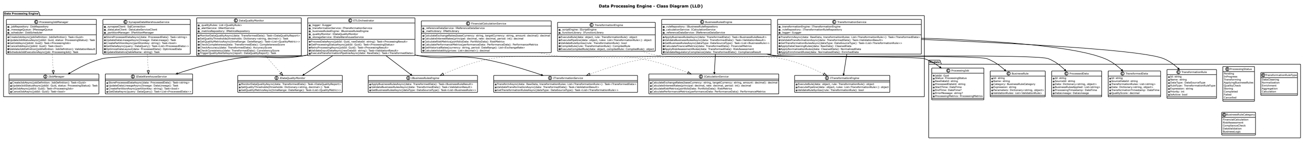
High-Level Design architecture diagram


FinHub Data Processing Solution - High Level Design (HLD)
BCRA API
Banco Central de la República Argentina - Exchange rates and monetary data
ARCA API
Financial transaction and settlement data
Market Data APIs
Stock prices, bond yields, commodity prices
Credit Bureau APIs
Credit scores and risk assessment data
Internal Systems
Customer transactions and account data
Exchange rate data [HTTPS/REST]
Settlement data [HTTPS/REST]
Market prices [HTTPS/REST]
Credit data [HTTPS/REST]
Transaction data [HTTPS/REST]
FinHub Data Processing Platform [system]
Data Ingestion Layer [container]
Ingestion Scheduler [Azure Logic Apps]
Manages data collection schedules and triggers
API Gateway [Azure API Management]
Central entry point for all external data sources
Trigger collection [Event]
Route requests [HTTPS]
Data Connectors [Azure Functions (C#)]
Specialized connectors for BCRA, ARCA, and other APIs
Validate data [Message Queue]
Data Validation Service [Azure Functions (C#)]
Validates data quality and format
Authenticate [OAuth2]
Store raw data [HTTPS]
Health metrics [Telemetry]
Trigger ETL [Event]
Data Processing Engine [container]
ETL Orchestrator [Azure Data Factory]
Manages the entire ETL workflow
Data Quality Monitor [Azure Monitor]
Monitors data quality and alerts
Transform data [Pipeline]
Monitor quality [Telemetry]
Data Transformation Service [Azure Synapse Analytics]
Performs data cleaning and enrichment
Check compliance [Policy]
Apply rules [Function call]
Log operations [Event]
Business Rules Engine [Azure Functions (C#)]
Applies financial business rules
Store processed data [HTTPS]
Data Storage Layer [container]
Archive Storage [Azure Blob Storage (Archive)]
Long-term compliance storage
Metadata Repository [Azure SQL Database]
Data lineage and schema information
Data Warehouse [Azure Synapse Analytics]
Stores cleaned and transformed data
Raw Data Lake [Azure Data Lake Storage Gen2]
Stores unprocessed data from various sources
Send alerts [Event]
Encrypt data [Service call]
Query data [SQL]
Backup data [Scheduled]
System Administrator
Manages and monitors the data platform
Monitors system [HTTPS]
Security & Compliance Layer [container]
Compliance Monitor [Azure Policy]
Ensures regulatory compliance
Audit Logger [Azure Monitor]
Tracks all data access and modifications
Encryption Service [Azure Key Vault]
Data encryption management
Authentication Service [Azure Active Directory]
User authentication and authorization
Analytics & Reporting Layer [container]
Analytics Engine [Azure Synapse Analytics]
Complex financial calculations
Infrastructure & Operations [container]
Backup & Recovery [Azure Backup]
Data backup and disaster recovery
Configuration Manager [Azure App Configuration]
Centralized configuration
Alert Manager [Azure Monitor]
Manages alerts and notifications
Monitoring Service [Azure Monitor]
System health and performance
Financial Analyst
Analyzes financial data and creates reports
Management Team
Reviews dashboards and makes strategic decisions
Views analytics [HTTPS]
Real-time data [API]
Expose APIs [REST]
Analytics results [API]
Reviews reports [HTTPS]
Dashboard Service [Power BI]
Real-time management dashboards
Analytics API [Azure Functions (C#)]
Exposes analytics data via REST APIs
Report Generator [Power BI Service]
Scheduled and on-demand reports
Legend
person
system
container
external person
external system
external container

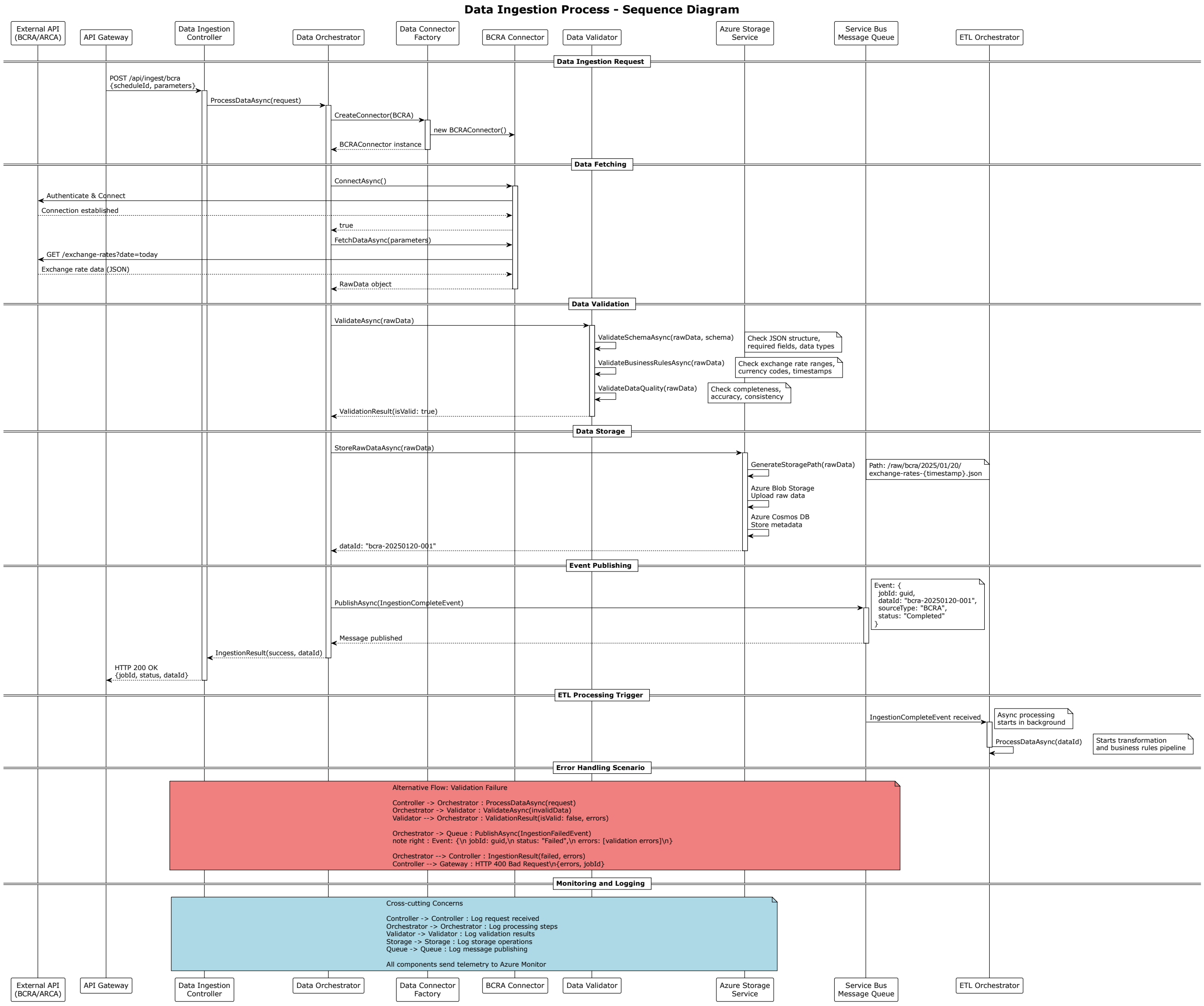
Data Ingestion Service class diagram


Data Ingestion Service - Class Diagram (LLD)
Data Ingestion Service
MarketDataConnector
_httpClient: HttpClient
_apiKey: string
_baseUrl: string
ConnectAsync(): Task<bool>
FetchDataAsync(parameters: FetchParameters): Task<RawData>
DisconnectAsync(): Task
ValidateConnectionAsync(): Task<bool>
GetMarketDataAsync(symbols: List<string>): Task<MarketData>
ARCAConnector
_httpClient: HttpClient
_credentials: ARCACredentials
_baseUrl: string
ConnectAsync(): Task<bool>
FetchDataAsync(parameters: FetchParameters): Task<RawData>
DisconnectAsync(): Task
ValidateConnectionAsync(): Task<bool>
AuthenticateAsync(): Task<string>
ParseARCAResponse(response: HttpResponseMessage): RawData
BCRAConnector
_httpClient: HttpClient
_apiKey: string
_baseUrl: string
ConnectAsync(): Task<bool>
FetchDataAsync(parameters: FetchParameters): Task<RawData>
DisconnectAsync(): Task
ValidateConnectionAsync(): Task<bool>
BuildRequestUrl(parameters: FetchParameters): string
ParseBCRAResponse(response: HttpResponseMessage): RawData
IDataConnector
ConnectAsync(): Task<bool>
FetchDataAsync(parameters: FetchParameters): Task<RawData>
DisconnectAsync(): Task
ValidateConnectionAsync(): Task<bool>
DataConnectorFactory
_serviceProvider: IServiceProvider
_configuration: IConfiguration
CreateConnector(sourceType: DataSourceType): IDataConnector
GetSupportedSources(): List<DataSourceType>
IDataConnectorFactory
CreateConnector(sourceType: DataSourceType): IDataConnector
GetSupportedSources(): List<DataSourceType>
ServiceBusMessageQueue
_serviceBusClient: ServiceBusClient
_senderCache: ConcurrentDictionary<string, ServiceBusSender>
PublishAsync<T>(message: T, topic: string): Task
SubscribeAsync<T>(topic: string, handler: Func<T, Task>): Task
GetOrCreateSender(topic: string): ServiceBusSender
IMessageQueue
PublishAsync<T>(message: T, topic: string): Task
SubscribeAsync<T>(topic: string, handler: Func<T, Task>): Task
AzureStorageService
_blobServiceClient: BlobServiceClient
_cosmosClient: CosmosClient
_containerName: string
StoreRawDataAsync(data: RawData): Task<string>
StoreMetadataAsync(metadata: DataMetadata): Task<string>
GetRawDataAsync(id: string): Task<RawData>
GenerateStoragePath(data: RawData): string
IStorageService
StoreRawDataAsync(data: RawData): Task<string>
StoreMetadataAsync(metadata: DataMetadata): Task<string>
GetRawDataAsync(id: string): Task<RawData>
DataOrchestrator
_connectorFactory: IDataConnectorFactory
_validator: IDataValidator
_storageService: IStorageService
_messageQueue: IMessageQueue
ProcessDataAsync(request: IngestionRequest): Task<IngestionResult>
GetJobStatusAsync(jobId: Guid): Task<IngestionStatus>
CancelJobAsync(jobId: Guid): Task<bool>
ValidateAndStoreAsync(data: RawData): Task<ValidationResult>
PublishIngestionEventAsync(result: IngestionResult): Task
IDataOrchestrator
ProcessDataAsync(request: IngestionRequest): Task<IngestionResult>
GetJobStatusAsync(jobId: Guid): Task<IngestionStatus>
CancelJobAsync(jobId: Guid): Task<bool>
DataIngestionController
_logger: ILogger
_orchestrator: IDataOrchestrator
_validator: IDataValidator
IngestDataAsync(source: string, data: object): Task<IngestionResult>
GetIngestionStatusAsync(jobId: Guid): Task<IngestionStatus>
GetSupportedSourcesAsync(): Task<List<DataSource>>
DataValidator
_schemaRepository: ISchemaRepository
_businessRulesEngine: IBusinessRulesEngine
ValidateAsync(data: RawData): Task<ValidationResult>
ValidateSchemaAsync(data: RawData, schema: DataSchema): Task<SchemaValidationResult>
ValidateBusinessRulesAsync(data: RawData): Task<BusinessRuleValidationResult>
ValidateDataQuality(data: RawData): DataQualityResult
IDataValidator
ValidateAsync(data: RawData): Task<ValidationResult>
ValidateSchemaAsync(data: RawData, schema: DataSchema): Task<SchemaValidationResult>
ValidateBusinessRulesAsync(data: RawData): Task<BusinessRuleValidationResult>
Models
IngestionStatus
Pending
InProgress
Completed
Failed
Cancelled
DataSourceType
BCRA
ARCA
MarketData
CreditBureau
Internal
IngestionResult
JobId: Guid
Status: IngestionStatus
DataId: string
RecordsProcessed: int
Errors: List<string>
ProcessingTime: TimeSpan
IngestionRequest
JobId: Guid
SourceType: DataSourceType
Parameters: FetchParameters
Timestamp: DateTime
RequestedBy: string
ValidationResult
IsValid: bool
Errors: List<ValidationError>
Warnings: List<ValidationWarning>
ValidationTimestamp: DateTime
RawData
Id: string
SourceType: DataSourceType
Content: byte[]
Format: DataFormat
Timestamp: DateTime
Metadata: Dictionary<string, object>

Data Processing Engine class diagram


Data Processing Engine - Class Diagram (LLD)
Data Processing Engine
ProcessingJobManager
_jobRepository: IJobRepository
_messageQueue: IMessageQueue
_scheduler: IJobScheduler
CreateJobAsync(jobDefinition: JobDefinition): Task<Guid>
UpdateJobStatusAsync(jobId: Guid, status: ProcessingStatus): Task
GetJobAsync(jobId: Guid): Task<ProcessingJob>
CancelJobAsync(jobId: Guid): Task<bool>
ValidateJobDefinition(jobDefinition: JobDefinition): ValidationResult
ScheduleJobExecutionAsync(job: ProcessingJob): Task
SynapseDataWarehouseService
_synapseClient: SqlConnection
_dataLakeClient: DataLakeServiceClient
_partitionManager: IPartitionManager
StoreProcessedDataAsync(data: ProcessedData): Task<string>
UpdateDataLineageAsync(lineage: DataLineage): Task
CreatePartitionAsync(partitionKey: string): Task<bool>
GetDataAsync(query: DataQuery): Task<List<ProcessedData>>
OptimizeDataLayout(data: ProcessedData): OptimizedData
UpdateStatistics(tableName: string): Task
DataQualityMonitor
_qualityRules: List<QualityRule>
_alertService: IAlertService
_metricsRepository: IMetricsRepository
MonitorDataQualityAsync(data: TransformedData): Task<DataQualityReport>
SetQualityThresholds(thresholds: Dictionary<string, decimal>): Task
GetQualityMetricsAsync(timeRange: DateRange): Task<List<QualityMetric>>
CheckCompleteness(data: TransformedData): CompletenessScore
CheckAccuracy(data: TransformedData): AccuracyScore
CheckConsistency(data: TransformedData): ConsistencyScore
TriggerQualityAlertsAsync(report: DataQualityReport): Task
ETLOrchestrator
_logger: ILogger
_transformationService: ITransformationService
_businessRulesEngine: IBusinessRulesEngine
_qualityMonitor: IDataQualityMonitor
_storageService: IDataWarehouseService
ProcessDataAsync(jobId: Guid, rawDataId: string): Task<ProcessingResult>
GetProcessingStatusAsync(jobId: Guid): Task<ProcessingStatus>
RetryProcessingAsync(jobId: Guid): Task<ProcessingResult>
ValidateInputDataAsync(rawDataId: string): Task<ValidationResult>
ExecuteTransformationPipelineAsync(data: RawData): Task<TransformedData>
FinancialCalculationService
_referenceDataService: IReferenceDataService
_mathLibrary: IMathLibrary
CalculateExchangeRates(baseCurrency: string, targetCurrency: string, amount: decimal): decimal
CalculateInterestRates(principal: decimal, rate: decimal, period: int): decimal
CalculateRiskMetrics(portfolioData: PortfolioData): RiskMetrics
CalculatePerformanceMetrics(performanceData: PerformanceData): PerformanceMetrics
GetHistoricalRates(currency: string, period: DateRange): List<ExchangeRate>
CalculateVolatility(prices: List<decimal>): decimal
TransformationEngine
_scriptEngine: IScriptEngine
_functionLibrary: IFunctionLibrary
ExecuteRule(data: object, rule: TransformationRule): object
ExecutePipeline(data: object, rules: List<TransformationRule>): object
ValidateRuleSyntax(rule: TransformationRule): bool
CompileRule(rule: TransformationRule): CompiledRule
ExecuteCompiledRule(data: object, compiledRule: CompiledRule): object
BusinessRulesEngine
_ruleRepository: IBusinessRuleRepository
_calculationService: ICalculationService
_referenceDataService: IReferenceDataService
ApplyBusinessRulesAsync(data: TransformedData): Task<BusinessRuleResult>
ValidateBusinessRulesAsync(data: TransformedData): Task<ValidationResult>
GetBusinessRulesAsync(dataType: DataSourceType): Task<List<BusinessRule>>
CalculateFinancialMetrics(data: TransformedData): FinancialMetrics
ApplyRiskAssessmentRules(data: TransformedData): RiskAssessment
ValidateRegulatoryCompliance(data: TransformedData): ComplianceResult
TransformationService
_transformationEngine: ITransformationEngine
_ruleRepository: ITransformationRuleRepository
_logger: ILogger
TransformAsync(data: RawData, transformationRules: List<TransformationRule>): Task<TransformedData>
ValidateTransformationAsync(data: TransformedData): Task<ValidationResult>
GetTransformationRulesAsync(dataType: DataSourceType): Task<List<TransformationRule>>
ApplyDataCleaningRules(data: RawData): CleanedData
ApplyNormalizationRules(data: CleanedData): NormalizedData
ApplyEnrichmentRules(data: NormalizedData): EnrichedData
IJobManager
CreateJobAsync(jobDefinition: JobDefinition): Task<Guid>
UpdateJobStatusAsync(jobId: Guid, status: ProcessingStatus): Task
GetJobAsync(jobId: Guid): Task<ProcessingJob>
CancelJobAsync(jobId: Guid): Task<bool>
IDataWarehouseService
StoreProcessedDataAsync(data: ProcessedData): Task<string>
UpdateDataLineageAsync(lineage: DataLineage): Task
CreatePartitionAsync(partitionKey: string): Task<bool>
GetDataAsync(query: DataQuery): Task<List<ProcessedData>>
IDataQualityMonitor
MonitorDataQualityAsync(data: TransformedData): Task<DataQualityReport>
SetQualityThresholds(thresholds: Dictionary<string, decimal>): Task
GetQualityMetricsAsync(timeRange: DateRange): Task<List<QualityMetric>>
IBusinessRulesEngine
ApplyBusinessRulesAsync(data: TransformedData): Task<BusinessRuleResult>
ValidateBusinessRulesAsync(data: TransformedData): Task<ValidationResult>
GetBusinessRulesAsync(dataType: DataSourceType): Task<List<BusinessRule>>
ITransformationService
TransformAsync(data: RawData, transformationRules: List<TransformationRule>): Task<TransformedData>
ValidateTransformationAsync(data: TransformedData): Task<ValidationResult>
GetTransformationRulesAsync(dataType: DataSourceType): Task<List<TransformationRule>>
ICalculationService
CalculateExchangeRates(baseCurrency: string, targetCurrency: string, amount: decimal): decimal
CalculateInterestRates(principal: decimal, rate: decimal, period: int): decimal
CalculateRiskMetrics(portfolioData: PortfolioData): RiskMetrics
CalculatePerformanceMetrics(performanceData: PerformanceData): PerformanceMetrics
ITransformationEngine
ExecuteRule(data: object, rule: TransformationRule): object
ExecutePipeline(data: object, rules: List<TransformationRule>): object
ValidateRuleSyntax(rule: TransformationRule): bool
Models
ProcessingJob
JobId: Guid
Status: ProcessingStatus
RawDataId: string
ProcessedDataId: string
StartTime: DateTime
EndTime: DateTime?
ErrorMessage: string?
ProcessingMetrics: ProcessingMetrics
BusinessRule
Id: string
Name: string
Category: BusinessRuleCategory
Expression: string
Parameters: Dictionary<string, object>
ValidationRules: List<ValidationRule>
ProcessedData
Id: string
SourceId: string
Data: Dictionary<string, object>
BusinessRulesApplied: List<string>
ProcessingTimestamp: DateTime
DataLineage: DataLineage
TransformedData
Id: string
SourceDataId: string
TransformationRules: List<string>
Data: Dictionary<string, object>
TransformationTimestamp: DateTime
QualityScore: decimal
TransformationRule
Id: string
Name: string
DataType: DataSourceType
RuleType: TransformationRuleType
Expression: string
Priority: int
IsActive: bool
ProcessingStatus
Pending
InProgress
Transforming
ApplyingBusinessRules
QualityCheck
Storing
Completed
Failed
Cancelled
TransformationRuleType
DataCleaning
Normalization
Enrichment
Aggregation
Calculation
BusinessRuleCategory
FinancialCalculation
RiskAssessment
ComplianceCheck
DataValidation
BusinessLogic

Data ingestion sequence diagram

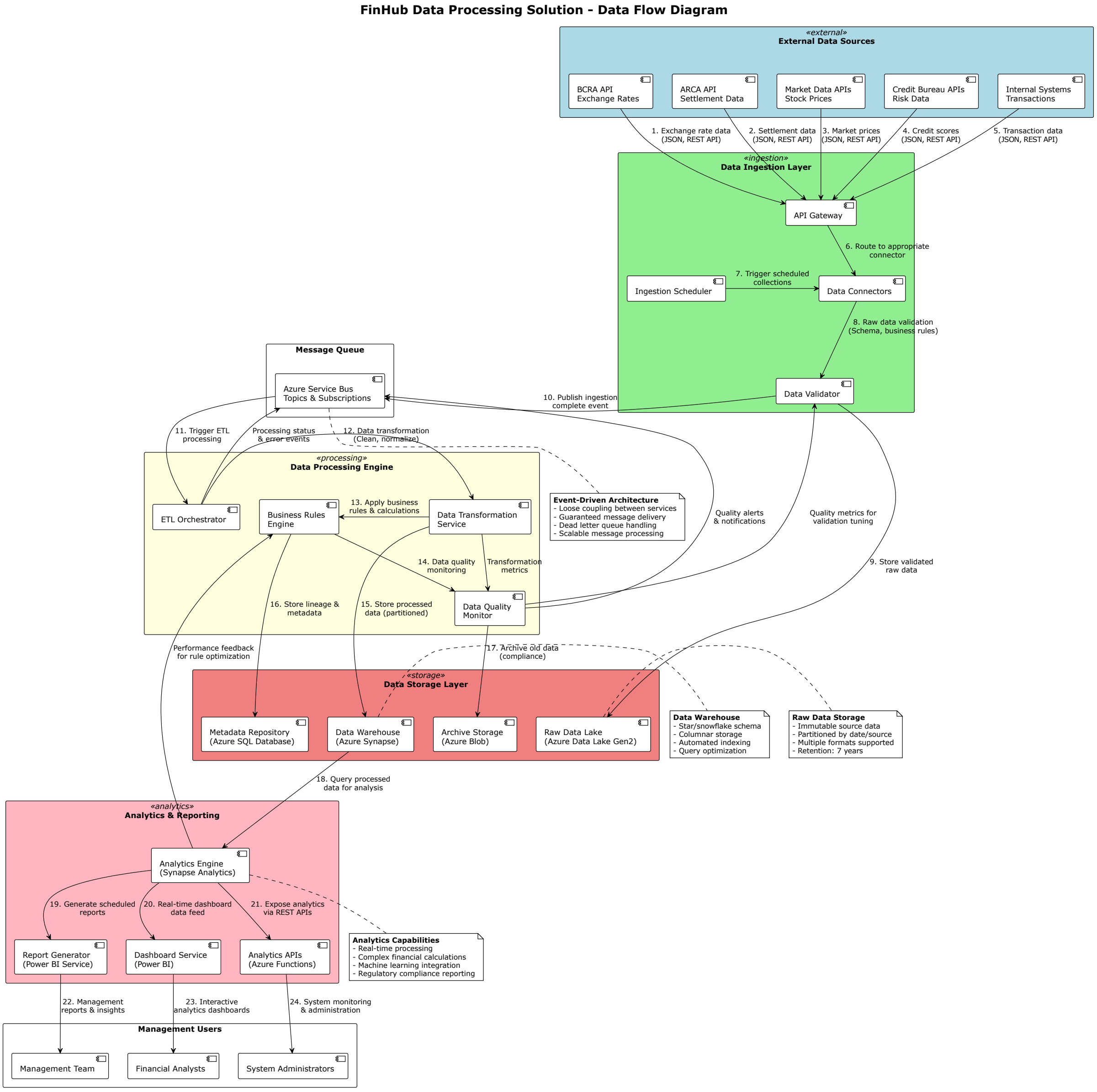

Data Ingestion Process - Sequence Diagram
External API (BCRA/ARCA)
API Gateway
Data Ingestion Controller
Data Orchestrator
Data Connector Factory
BCRA Connector
Data Validator
Azure Storage Service
Service Bus Message Queue
ETL Orchestrator
Data Ingestion Request
POST /api/ingest/bcra {scheduleId, parameters}
ProcessDataAsync(request)
CreateConnector(BCRA)
new BCRAConnector()
BCRAConnector instance
Data Fetching
ConnectAsync()
Authenticate & Connect
Connection established
true
FetchDataAsync(parameters)
GET /exchange-rates?date=today
Exchange rate data (JSON)
RawData object
Data Validation
ValidateAsync(rawData)
ValidateSchemaAsync(rawData, schema)
Check JSON structure, required fields, data types
ValidateBusinessRulesAsync(rawData)
Check exchange rate ranges, currency codes, timestamps
ValidateDataQuality(rawData)
Check completeness, accuracy, consistency
ValidationResult(isValid: true)
Data Storage
StoreRawDataAsync(rawData)
GenerateStoragePath(rawData)
Path: /raw/bcra/2025/01/20/ exchange-rates-{timestamp}.json
Azure Blob Storage Upload raw data
Azure Cosmos DB Store metadata
dataId: "bcra-20250120-001"
Event Publishing
PublishAsync(IngestionCompleteEvent)
Message published
IngestionResult(success, dataId)
HTTP 200 OK {jobId, status, dataId}
ETL Processing Trigger
IngestionCompleteEvent received
Async processing starts in background
ProcessDataAsync(dataId)
Starts transformation and business rules pipeline
Error Handling Scenario
Monitoring and Logging

```
Event: {
  jobId: guid,
  dataId: "bcra-20250120-001",
  sourceType: "BCRA",
  status: "Completed"
}
```

```
Alternative Flow: Validation Failure

Controller -> Orchestrator : ProcessDataAsync(request)
Orchestrator -> Validator : ValidateAsync(invalidData)
Validator --> Orchestrator : ValidationResult(isValid: false, errors)

Orchestrator -> Queue : PublishAsync(IngestionFailedEvent)
note right : Event: {\n jobId: guid,\n status: "Failed",\n errors: [validation errors]\n}

Orchestrator --> Controller : IngestionResult(failed, errors)
Controller --> Gateway : HTTP 400 Bad Request\n{errors, jobId}
```

```
Cross-cutting Concerns

Controller -> Controller : Log request received
Orchestrator -> Orchestrator : Log processing steps
Validator -> Validator : Log validation results
Storage -> Storage : Log storage operations
Queue -> Queue : Log message publishing

All components send telemetry to Azure Monitor
```

Data flow diagram



System Architecture Design Approach and Decisions

Design Summary

I built both solutions to be cloud-first, scalable, and secure—leaning heavily on Microsoft Azure's native tools. For the document classifier, I used C4 modeling to break down the architecture. For the FinHub pipeline, I went with a clean, layered approach powered by microservices and event-driven logic.

Architectural Highlights

- **Cloud-Native by Default**: Azure Functions and Synapse reduce overhead and handle scale with ease.
- **Event-Driven Messaging**: Service Bus keeps components loosely coupled and responsive.
- **Built-In Security**: Active Directory, Key Vault, and API Management ensure protection and compliance.

Technical Decisions

I went with C# for strong tooling and Azure integration. Architecture follows microservices with patterns like CQRS and event sourcing. AI features combine Azure Cognitive Services with custom models for flexibility and fast delivery.