Technical Specifications

Veria

1. INTRODUCTION

1.1 EXECUTIVE SUMMARY

1.1.1 Project Overview

Veria is a compliance middleware platform designed to bridge the gap between traditional finance and blockchain technology for tokenized real-world assets (RWAs). Positioned as "Plaid for tokenized funds," Veria provides comprehensive compliance infrastructure for the tokenization lifecycle, starting with US Treasuries and Money Market Funds. The platform operates as a pnpm monorepo implementing microservices architecture with 11 specialized TypeScript services orchestrated through an API Gateway pattern.

1.1.2 Core Business Problem

The tokenized real-world assets market faces a critical infrastructure gap: the lack of integrated compliance middleware to connect traditional finance systems with blockchain technology. Current solutions require manual compliance processes, fragmented KYC/AML procedures, and complex regulatory reporting mechanisms that create barriers to adoption and increase operational risk. Veria addresses this by providing automated compliance workflows, standardized asset onboarding, and audit-ready reporting capabilities.

1.1.3 Key Stakeholders and Users

Stakeholder Group	Primary Role	Key Interactions
Compliance Officers	Primary users for asset onboarding and regulatory reporting	Asset creation, document management, audit trail generation
Asset Issuers & Fund Managers	Securities tokenization and distribution	Asset configuration, investor approval workflows
Investors	Asset acquisition and portfolio management	KYC/KYB submission, credential management, transaction execution
SMB Finance Teams & CPAs	Adoption wedge and market entry	QuickBooks/Xero integration, accounting reconciliation

Additional stakeholders include KYC/KYB providers (Chainalysis, TRM Labs, Jumio, Onfido), regulatory bodies, auditors, and custody providers such as BNY Mellon.

1.1.4 Expected Business Impact and Value Proposition

Veria delivers measurable business value through:

- **Operational Efficiency**: Reduces time to first transaction and investor onboarding time through automated compliance workflows
- **Risk Mitigation**: Ensures regulatory compliance at every step with built-in screening and rules engines
- **Audit Readiness**: Provides comprehensive compliance export capabilities with manifest-driven evidence collection
- **Market Access**: Bridges traditional finance and DeFi ecosystems, enabling broader asset distribution
- **Scalability**: Supports multiple jurisdictions, asset types, and integration patterns for enterprise growth

1.2 SYSTEM OVERVIEW

1.2.1 Project Context

1.2.1.1 Business Context and Market Positioning

Veria operates in the emerging tokenized assets market, specifically targeting the compliance and distribution middleware layer. The platform addresses the growing need for compliant tokenization infrastructure as traditional financial institutions explore blockchain-based asset distribution. With an initial focus on US Treasuries and Money Market Funds, Veria provides the regulatory foundation necessary for institutional adoption of tokenized securities.

1.2.1.2 Current System Limitations

Existing tokenization platforms lack integrated compliance middleware, requiring manual processes for:

- Asset onboarding and regulatory documentation
- Investor eligibility verification and KYC/AML screening
- Regulatory reporting and audit trail generation
- Multi-provider integration for compliance services

1.2.1.3 Integration with Enterprise Landscape

Veria integrates with existing enterprise systems through:

- QuickBooks/Xero connectors for SMB finance teams
- Multi-provider KYC integration (Chainalysis, TRM Labs, Jumio, Onfido)
- Blockchain network support (Ethereum, Polygon, Solana)
- Custody provider integration (BNY Mellon and others)
- Payment processing through Stripe integration

1.2.2 High-Level Description

1.2.2.1 Primary System Capabilities

Veria implements three core workflow capabilities:

Asset Onboarding Flow: Complete lifecycle management from asset entry creation through regulatory validation, including custody provider configuration, SPV/Trust structure setup, tokenization parameter definition, and jurisdiction rule compliance.

Investor Management Flow: Comprehensive investor registry management with KYC/KYB document processing, eligibility verification for accredited and qualified purchaser status, approval workflows with audit trails, and access credential issuance.

Compliance Export Flow: Audit-ready evidence collection with period-based selection, comprehensive evidence gathering, ZIP generation with manifest.json, signed URL creation for secure download, and complete audit trail logging.

1.2.2.2 Major System Components

Component Category	Services	Completion Status
Backend Services	Gateway (90%), Identity (30%), Policy (40%), Compliance (0%), Audit Log Writer (35%)	~35% average
Frontend Applications	Compliance Dashboard (React/Vite, 10%), Frontend (Next.js, minimal)	~5% average
Shared Packages	auth-middleware, database (Prisma), sdk-ts, blockchain, compliance_middleware	~30% average
Smart Contracts	VeriaSecurityToken (ERC-3643), ModularCompliance, IdentityRegistry	0% complete

1.2.2.3 Core Technical Approach

The system implements a service mesh architecture with:

- **API Gateway Pattern**: All backend services (ports 4001-4005) accessed through Gateway service on port 4000
- **RESTful API Design**: JWT-based authentication with role-based access control (RBAC)
- **Event-Driven Architecture**: Real-time compliance monitoring and screening
- **Multi-Provider Integration**: Standardized interfaces for KYC/AML and blockchain providers
- **YAML-Driven Configuration**: Declarative compliance rules and policy management

1.2.3 Success Criteria

1.2.3.1 Measurable Objectives

Metric Category	Target	Measurement Method
Performance	API response time p99 < 200ms, System uptime 99.99% SLA	Prometheus/Grafana monitoring
Scalability	Transaction throughput 10,000+ TPS	k6 performance testing
Quality	Test coverage >80% services, > 70% frontend	Automated testing pipeline

1.2.3.2 Critical Success Factors

- **Compliance Service Completion**: Currently at 0% and identified as critical blocker
- **Database Optimization**: Early performance optimization for scale requirements

- **Security Audit**: No critical vulnerabilities in external security assessment
- **Integration Testing**: Multi-provider KYC and blockchain integration validation

1.2.3.3 Key Performance Indicators

Business KPIs include:

- Time to first transaction reduction
- Investor onboarding time minimization
- Compliance export generation time < 5 minutes
- Multi-jurisdiction support capability
- Major KYC provider integration coverage

1.3 SCOPE

1.3.1 In-Scope Elements

1.3.1.1 Core Features and Functionalities

Must-Have Capabilities:

- Complete asset onboarding workflow with custody provider configuration
- Investor management with KYC/KYB document processing and eligibility verification
- Compliance export with audit-ready evidence collection and manifest generation
- JWT-based authentication with role-based access control
- Multi-provider KYC integration with standardized interfaces

Primary User Workflows:

- Asset creation and tokenization parameter configuration

- Investor registry management and approval processes
- Regulatory document attachment and validation
- Treasury operations including deposit and withdrawal
- DeFi integration for staking and liquidity provision

Essential Integrations:

- Multi-chain blockchain support (Ethereum, Polygon, Solana initially)
- KYC provider APIs (Chainalysis, TRM Labs, Jumio, Onfido)
- Payment processing through Stripe
- Planned QuickBooks/Xero connector for SMB adoption

1.3.1.2 Implementation Boundaries

Boundary Type	Coverage	Specifications
System Boundaries	Microservices architecture with 11 specialized services	Gateway-mediated service mesh pattern
User Groups	Compliance officers, asset issuers, investors, SMB finance teams	Role-based access control implementation
Geographic Coverage	US market initially	US Treasuries and Money Market Funds focus
Data Domains	Asset metadata, investor records, compliance documentation	PostgreSQL with Prisma ORM, UUID primary keys

1.3.1.3 Technical Requirements

Core Technical Capabilities:

- Real-time compliance screening and rules engine
- Dual-write audit logging pattern for complete audit trails
- Multi-signature wallet support for enhanced security
- Hardware Security Module (HSM) integration planning
- Circuit breakers for service dependency management

1.3.2 Out-of-Scope Elements

1.3.2.1 Explicitly Excluded Features

UI/UX Exclusions:

- Graph visualization libraries or interfaces
- Timeline interfaces or custom IDE components
- Drawing tools or visual editors
- Vislzr integration components

Development Environment Exclusions:

- Real KYC provider APIs (development uses mocked implementations)
- Actual blockchain integration (development uses mocked services)
- Production blockchain deployment in initial phase

1.3.2.2 Future Phase Considerations

Phase 2 Capabilities:

- Advanced AI-powered compliance features through AI Broker service
- Mobile native applications for investor access
- International market expansion beyond US jurisdiction
- Graph visualization capabilities through Graph Service

Long-Term Roadmap:

- Production blockchain deployment with full smart contract audit
- Complex compliance rule visualization and management interfaces
- Advanced timeline interfaces for compliance workflow tracking
- Custom IDE components for compliance rule development

1.3.2.3 Integration Points Not Covered

Excluded Integrations:

- Real-time blockchain event monitoring in development phase
- Production-grade custody provider integrations (BNY Mellon integration planned)
- Advanced DeFi protocol integrations beyond basic staking
- International KYC provider networks

References

Files Examined:

- `README.md` - Project overview and technology stack documentation
- `PRD.md` - Master product requirements index and business context
- `BLITZY_SETUP.md` - Development setup procedures and deployment configuration
- `package.json` - Root monorepo configuration and dependency management
- `.env.example` - Environment configuration template and service port allocation

Folders Analyzed:

- `services/` - Microservices architecture with 11 specialized TypeScript services
- `packages/` - Shared infrastructure including auth-middleware, database, and sdk-ts
- `apps/` - Frontend applications including compliance dashboard and Next.js frontend
- `contracts/` - Smart contract architecture with ERC-3643 compliant tokens
- `docs/` - Comprehensive documentation including architecture, roadmap, and sprint plans

Documentation Sources:

- `docs/ARCHITECTURE.md` - Canonical architecture reference and service specifications

- `docs/PRODUCT_REQUIREMENTS.md` - Comprehensive PRD v3.0 with business requirements
- `docs/ROADMAP.md` - 15-week development roadmap to MVP delivery
- `docs/IMPLEMENTATION_STATUS.md` - Current progress tracking across all components
- `docs/API.md` - RESTful API endpoint documentation and authentication specifications

2. PRODUCT REQUIREMENTS

2.1 FEATURE CATALOG

2.1.1 Asset Management Features

2.1.1.1 Asset Tokenization (F-001)

Feature Metadata:

- **Feature ID**: F-001
- **Feature Name**: Asset Tokenization and Configuration
- **Feature Category**: Core Asset Management
- **Priority Level**: Critical
- **Status**: Proposed

Description:

- **Overview**: Complete lifecycle management for tokenizing real-world assets, starting with US Treasuries and Money Market Funds, including custody provider configuration and ERC-3643 compliant token deployment
- **Business Value**: Enables compliant digital asset creation with regulatory alignment, reducing time-to-market from months to days

- **User Benefits**: Streamlined asset onboarding, automated compliance verification, standardized tokenization process
- **Technical Context**: Integration with custody providers (BNY Mellon), SPV/Trust structure setup, multi-chain deployment capability

Dependencies:

- **Prerequisite Features**: Organization Registry (F-009), Compliance Rules Engine (F-005)
- **System Dependencies**: PostgreSQL database, Smart Contract infrastructure, Blockchain service
- **External Dependencies**: Custody provider APIs, Blockchain networks (Ethereum, Polygon, Solana)
- **Integration Requirements**: Multi-chain deployment capability, custody provider webhooks, smart contract audit

2.1.1.2 Asset Document Management (F-002)

Feature Metadata:

- **Feature ID**: F-002
- **Feature Name**: Product Documentation System
- **Feature Category**: Asset Management
- **Priority Level**: High
- **Status**: Proposed

Description:

- **Overview**: Manage regulatory documents and disclosures for tokenized assets with version control and audit trail
- **Business Value**: Ensures regulatory compliance and transparency, reduces audit preparation time
- **User Benefits**: Centralized document repository, automated document verification, access control management
- **Technical Context**: File storage with hash verification, metadata tracking, signed URLs for secure access

Dependencies:

- **Prerequisite Features**: Asset Tokenization (F-001)
- **System Dependencies**: Document storage system, PostgreSQL for metadata
- **External Dependencies**: Cloud storage providers (AWS S3)
- **Integration Requirements**: File upload APIs, document verification services, audit logging

2.1.2 Compliance and Verification Features

2.1.2.1 Multi-Provider KYC Integration (F-003)

Feature Metadata:

- **Feature ID**: F-003
- **Feature Name**: KYC/KYB Verification System
- **Feature Category**: Compliance
- **Priority Level**: Critical
- **Status**: In Development (30% complete)

Description:

- **Overview**: Multi-provider KYC/KYB verification with fallback capabilities and orchestration logic
- **Business Value**: Ensures investor eligibility and regulatory compliance, reduces verification failures through redundancy
- **User Benefits**: Fast verification processing, multiple provider options, high availability verification
- **Technical Context**: Integration with Chainalysis, TRM Labs, Jumio, Onfido with standardized interfaces

Dependencies:

- **Prerequisite Features**: User Management (F-010)

- **System Dependencies**: Redis for caching, PostgreSQL for persistence, KYC Provider service
- **External Dependencies**: KYC provider APIs
- **Integration Requirements**: Webhook handlers, provider adapters, fallback orchestration logic

2.1.2.2 AML Screening and Monitoring (F-004)

Feature Metadata:

- **Feature ID**: F-004
- **Feature Name**: Anti-Money Laundering System
- **Feature Category**: Compliance
- **Priority Level**: Critical
- **Status**: Proposed

Description:

- **Overview**: Real-time AML screening and continuous monitoring with sanctions list checking
- **Business Value**: Prevents financial crime and ensures regulatory compliance, reduces regulatory risk
- **User Benefits**: Automated screening processes, real-time risk scoring, comprehensive alert management
- **Technical Context**: Transaction monitoring, sanctions database integration, risk assessment algorithms

Dependencies:

- **Prerequisite Features**: KYC Integration (F-003), User Management (F-010)
- **System Dependencies**: Real-time processing engine, alert management system
- **External Dependencies**: Sanctions databases (OFAC, EU sanctions), AML service providers

- **Integration Requirements**: Real-time data feeds, alert routing system, compliance reporting

2.1.2.3 Compliance Rules Engine (F-005)

Feature Metadata:

- **Feature ID**: F-005
- **Feature Name**: Dynamic Compliance Policy System
- **Feature Category**: Compliance
- **Priority Level**: Critical
- **Status**: Proposed (0% complete)

Description:

- **Overview**: YAML-driven compliance rules engine with jurisdiction-specific logic and decision tracking
- **Business Value**: Automated compliance verification across jurisdictions, reduces manual review overhead
- **User Benefits**: Configurable compliance rules, automated decision-making, complete audit trail
- **Technical Context**: Rules evaluation engine, policy versioning, decision logging, YAML configuration

Dependencies:

- **Prerequisite Features**: None (foundational service)
- **System Dependencies**: Policy service, Redis for caching, audit logging
- **External Dependencies**: None
- **Integration Requirements**: Gateway integration, all services for rule evaluation, audit system

2.1.3 Transaction and Treasury Features

2.1.3.1 Treasury Operations Management (F-006)

Feature Metadata:

- **Feature ID**: F-006
- **Feature Name**: Deposit and Withdrawal System
- **Feature Category**: Treasury
- **Priority Level**: High
- **Status**: Proposed

Description:

- **Overview**: Comprehensive treasury operations including fund deposits, withdrawals, and balance management
- **Business Value**: Enables capital flow management and liquidity operations, automates reconciliation
- **User Benefits**: Real-time balance tracking, transaction history, automated treasury reconciliation
- **Technical Context**: Multi-signature approvals, transaction batching, gas optimization, payment processing

Dependencies:

- **Prerequisite Features**: User Wallet Management (F-011), Asset Tokenization (F-001)
- **System Dependencies**: Blockchain service, transaction database, payment service
- **External Dependencies**: Payment processors (Stripe), blockchain networks
- **Integration Requirements**: Payment gateway integration, blockchain RPC connections, wallet management

2.1.3.2 Transaction Approval Workflow (F-007)

Feature Metadata:

- **Feature ID**: F-007
- **Feature Name**: Multi-Level Transaction Approval
- **Feature Category**: Transaction Management
- **Priority Level**: High
- **Status**: Proposed

Description:

- **Overview**: Configurable approval workflows for transactions with role-based authorization
- **Business Value**: Risk mitigation through controlled approvals, ensures proper authorization
- **User Benefits**: Flexible approval chains, transparent approval process, automated notifications
- **Technical Context**: Role-based approvals, threshold limits, automated routing, notification system

Dependencies:

- **Prerequisite Features**: User Roles and Permissions (F-012), Treasury Operations (F-006)
- **System Dependencies**: Workflow engine, notification service
- **External Dependencies**: Email/SMS notification services
- **Integration Requirements**: Role management integration, notification delivery systems

2.1.4 Reporting and Audit Features

2.1.4.1 Regulatory Reporting System (F-008)

Feature Metadata:

- **Feature ID**: F-008
- **Feature Name**: SAR/CTR Report Generation
- **Feature Category**: Reporting

- **Priority Level**: High
- **Status**: In Development (35% complete)

Description:

- **Overview**: Automated generation of Suspicious Activity Reports and Currency Transaction Reports
- **Business Value**: Ensures regulatory compliance, reduces manual reporting effort, ensures timely submission
- **User Benefits**: Automated report generation, multiple export formats, scheduled report delivery
- **Technical Context**: Template-based generation, PDF/Excel/JSON exports, job scheduling system

Dependencies:

- **Prerequisite Features**: Transaction Management, AML Monitoring (F-004)
- **System Dependencies**: Report generation service, job scheduler, file storage
- **External Dependencies**: None
- **Integration Requirements**: File storage systems, email delivery services

2.1.4.2 Compliance Export System (F-013)

Feature Metadata:

- **Feature ID**: F-013
- **Feature Name**: Audit Evidence Collection
- **Feature Category**: Audit
- **Priority Level**: High
- **Status**: In Development (35% complete)

Description:

- **Overview**: Comprehensive audit trail and evidence export system with manifest generation
- **Business Value**: Facilitates regulatory audits, reduces audit preparation time, ensures evidence completeness
- **User Benefits**: One-click export functionality, complete audit trail, secure evidence delivery
- **Technical Context**: Dual-write audit logging pattern, ZIP generation with manifest, signed URLs

Dependencies:

- **Prerequisite Features**: All transaction and compliance features
- **System Dependencies**: Audit log service, file storage system, ZIP generation
- **External Dependencies**: None
- **Integration Requirements**: All services must implement audit logging, file delivery system

2.1.5 User and Organization Management Features

2.1.5.1 Organization Registry (F-009)

Feature Metadata:

- **Feature ID**: F-009
- **Feature Name**: Organization Management System
- **Feature Category**: User Management
- **Priority Level**: Critical
- **Status**: Proposed

Description:

- **Overview**: Comprehensive organization management including issuers, distributors, and institutional investors

- **Business Value**: Enables B2B relationships and multi-tenant operations, supports institutional adoption
- **User Benefits**: Organization profile management, KYB status tracking, relationship management
- **Technical Context**: Multi-tenant architecture, organization types, jurisdiction tracking, relationship mapping

Dependencies:

- **Prerequisite Features**: None (foundational service)
- **System Dependencies**: Database, authentication system, identity service
- **External Dependencies**: None
- **Integration Requirements**: KYB provider integration, user management integration

2.1.5.2 User Management System (F-010)

Feature Metadata:

- **Feature ID**: F-010
- **Feature Name**: User Registry and Profile Management
- **Feature Category**: User Management
- **Priority Level**: Critical
- **Status**: In Development (30% complete)

Description:

- **Overview**: Complete user lifecycle management with profile management and credential tracking
- **Business Value**: Enables user onboarding and access control, supports scalable user management
- **User Benefits**: Self-service profile management, credential issuance, session management
- **Technical Context**: JWT authentication, session management, WebAuthn support, role-based access

Dependencies:

- **Prerequisite Features**: Organization Registry (F-009)
- **System Dependencies**: Identity service, Redis for sessions, database
- **External Dependencies**: None
- **Integration Requirements**: Email verification services, SMS verification, authentication providers

2.1.5.3 User Wallet Management (F-011)

Feature Metadata:

- **Feature ID**: F-011
- **Feature Name**: Wallet and Address Management
- **Feature Category**: User Management
- **Priority Level**: High
- **Status**: Proposed

Description:

- **Overview**: User wallet address management with multi-chain support and security features
- **Business Value**: Enables secure token transactions and multi-chain operations
- **User Benefits**: Multi-chain wallet support, secure address validation, transaction history
- **Technical Context**: Address validation, multi-chain support, wallet linking, security verification

Dependencies:

- **Prerequisite Features**: User Management (F-010)
- **System Dependencies**: Blockchain service, wallet verification
- **External Dependencies**: Blockchain networks
- **Integration Requirements**: Multi-chain RPC connections, wallet verification services

2.1.5.4 User Roles and Permissions (F-012)

Feature Metadata:

- **Feature ID**: F-012
- **Feature Name**: Role-Based Access Control System
- **Feature Category**: User Management
- **Priority Level**: Critical
- **Status**: Proposed

Description:

- **Overview**: Comprehensive RBAC system with granular permissions and organization-based access
- **Business Value**: Ensures proper access control and security, enables fine-grained authorization
- **User Benefits**: Role-based access, granular permissions, secure resource access
- **Technical Context**: RBAC implementation, permission matrix, organization-scoped access, JWT claims

Dependencies:

- **Prerequisite Features**: User Management (F-010), Organization Registry (F-009)
- **System Dependencies**: Authentication middleware, database
- **External Dependencies**: None
- **Integration Requirements**: All services for permission enforcement

2.1.6 Infrastructure and Platform Features

2.1.6.1 Smart Contract Management (F-014)

Feature Metadata:

- **Feature ID**: F-014

- **Feature Name**: Smart Contract Deployment and Management
- **Feature Category**: Infrastructure
- **Priority Level**: Critical
- **Status**: Proposed

Description:

- **Overview**: ERC-3643 compliant smart contract deployment and management system
- **Business Value**: Enables compliant tokenization with regulatory features built-in
- **User Benefits**: Automated contract deployment, compliance features, upgrade management
- **Technical Context**: ERC-3643 implementation, modular compliance, identity registry integration

Dependencies:

- **Prerequisite Features**: Asset Tokenization (F-001), Compliance Rules Engine (F-005)
- **System Dependencies**: Blockchain service, smart contract templates
- **External Dependencies**: Blockchain networks, gas fee management
- **Integration Requirements**: Multi-chain deployment, contract verification, upgrade mechanisms

2.1.6.2 Blockchain Service Integration (F-015)

Feature Metadata:

- **Feature ID**: F-015
- **Feature Name**: Multi-Chain Blockchain Integration
- **Feature Category**: Infrastructure
- **Priority Level**: Critical
- **Status**: Proposed

Description:

- **Overview**: Multi-chain blockchain integration with standardized interfaces
- **Business Value**: Enables multi-chain operations and blockchain abstraction
- **User Benefits**: Chain-agnostic operations, reliable blockchain connectivity, transaction management
- **Technical Context**: Multi-chain RPC management, transaction broadcasting, event monitoring

Dependencies:

- **Prerequisite Features**: Smart Contract Management (F-014)
- **System Dependencies**: Blockchain service, transaction queue
- **External Dependencies**: Blockchain network RPC endpoints
- **Integration Requirements**: All blockchain-dependent services

2.1.6.3 Holdings and Balance Tracking (F-016)

Feature Metadata:

- **Feature ID**: F-016
- **Feature Name**: Portfolio and Holdings Management
- **Feature Category**: Asset Management
- **Priority Level**: High
- **Status**: Proposed

Description:

- **Overview**: Real-time portfolio tracking and balance management across multiple assets
- **Business Value**: Provides accurate portfolio valuation and performance tracking
- **User Benefits**: Real-time balance updates, portfolio analytics, transaction history
- **Technical Context**: Real-time balance calculation, multi-asset support, historical tracking

Dependencies:

- **Prerequisite Features**: Asset Tokenization (F-001), Treasury Operations (F-006)
- **System Dependencies**: Database, blockchain service, calculation engine
- **External Dependencies**: Price feeds, blockchain data
- **Integration Requirements**: Price feed integration, blockchain event monitoring

2.2 FUNCTIONAL REQUIREMENTS TABLE

2.2.1 Asset Tokenization Requirements (F-001)

Requirement ID	Description	Acceptance Criteria	Priority	Complexity
F-001-RQ-001	Create new asset entry	Asset record created with all required metadata fields populated	Must-Have	Medium
F-001-RQ-002	Configure custody provider	Custody provider selected and API integration validated	Must-Have	High
F-001-RQ-003	Define tokenization parameters	Token supply, symbol, decimals, and compliance rules configured	Must-Have	Medium
F-001-RQ-004	Deploy smart contract	ERC-3643 compliant contract deployed to target blockchain	Must-Have	High

Requirement ID	Description	Acceptance Criteria	Priority	Complexity
F-001-RQ-005	Set compliance rules	Jurisdiction-specific compliance rules attached and validated	Must-Have	Medium
F-001-RQ-006	Attach regulatory documents	Required documents uploaded with hash verification	Should-Have	Low

Technical Specifications:

- **Input Parameters**: Asset metadata (name, type, jurisdiction), custody configuration, token parameters (symbol, supply, decimals), compliance rule IDs, regulatory documents
- **Output/Response**: Asset ID (UUID), token contract address, deployment transaction hash, compliance verification status
- **Performance Criteria**: Contract deployment completion < 60 seconds, metadata storage < 500ms, document upload < 2 seconds
- **Data Requirements**: Asset type enumeration, supported currencies, decimal precision (0-18), document type validation

Validation Rules:

- **Business Rules**: Minimum investment thresholds ($1,000-$250,000), maximum supply limits based on asset type
- **Data Validation**: Token symbol format (3-10 uppercase characters), positive supply values, valid jurisdiction codes
- **Security Requirements**: Multi-signature deployment approval, custody provider API key validation, document integrity verification
- **Compliance Requirements**: Asset type must match jurisdiction regulations, required documents per asset class

2.2.2 KYC Integration Requirements (F-003)

Requirement ID	Description	Acceptance Criteria	Priority	Complexity
F-003-RQ-001	Submit KYC documents	Documents uploaded securely with metadata tracking	Must-Have	Medium
F-003-RQ-002	Provider orchestration	Primary/fallback provider selection with load balancing	Must-Have	High
F-003-RQ-003	Verification result caching	Results cached for performance with 1-hour TTL	Should-Have	Low
F-003-RQ-004	Webhook processing	Provider webhooks processed with retry logic	Must-Have	Medium
F-003-RQ-005	Risk scoring calculation	Aggregate risk score computed from multiple providers	Must-Have	High
F-003-RQ-006	Status tracking	Real-time KYC status updates across all touchpoints	Must-Have	Medium

Technical Specifications:

- **Input Parameters**: User ID (UUID), document type (passport, driver_license, utility_bill), document binary data, provider preference
- **Output/Response**: Verification ID, status (pending, approved, rejected), risk score (0-100), provider used, expiry date
- **Performance Criteria**: Verification initiation response < 30 seconds, webhook processing < 5 seconds, status update propagation < 10 seconds
- **Data Requirements**: Supported document types, provider configuration, risk threshold levels (low: 0-30, medium: 31-70, high: 71-100)

Validation Rules:

- **Business Rules**: Maximum 3 verification attempts per user per 24 hours, risk scores above 70 require manual review
- **Data Validation**: Document size limits (5MB), supported file formats (JPG, PNG, PDF), minimum image resolution
- **Security Requirements**: Document encryption at rest, PII data masking, access logging for all document operations
- **Compliance Requirements**: GDPR compliance for EU residents, document retention policies, right to erasure implementation

2.2.3 Compliance Rules Engine Requirements (F-005)

Requirement ID	Description	Acceptance Criteria	Priority	Complexity
F-005-RQ-001	YAML rule definition	Rules defined in YAML format with validation schema	Must-Have	Medium
F-005-RQ-002	Real-time rule evaluation	Rules processed within performance SLA	Must-Have	High
F-005-RQ-003	Decision audit logging	All rule decisions logged with complete context	Must-Have	Medium
F-005-RQ-004	Rule versioning system	Version control for rule changes with rollback capability	Should-Have	Medium
F-005-RQ-005	Multi-jurisdiction support	Rules support multiple jurisdictions simultaneously	Must-Have	High

Requirement ID	Description	Acceptance Criteria	Priority	Complexity
F-005-RQ-006	Rule simulation testing	Test rules against historical data before activation	Could-Have	Medium

Technical Specifications:

- **Input Parameters**: User context (ID, jurisdiction, accreditation status), transaction details (amount, asset type), rule set version
- **Output/Response**: Decision (ALLOW, DENY, MANUAL_REVIEW), applicable rules list, decision reasoning, confidence score
- **Performance Criteria**: Rule evaluation < 100ms (p95), < 200ms (p99), concurrent evaluations > 1000 TPS
- **Data Requirements**: Rule conditions (AND, OR, NOT logic), action types, metadata for audit trail

Validation Rules:

- **Business Rules**: Rules must have valid effective date ranges, maximum rule complexity limits, mandatory approval for high-impact rules
- **Data Validation**: YAML schema validation, rule syntax verification, circular dependency detection
- **Security Requirements**: Rule tampering detection, approval workflow for rule changes, encrypted rule storage
- **Compliance Requirements**: Regulatory rule mapping, jurisdiction-specific validation, audit trail for rule modifications

2.2.4 Treasury Operations Requirements (F-006)

Requirement ID	Description	Acceptance Criteria	Priority	Complexity
F-006-RQ-001	Process fund deposits	Fiat and crypto deposits processed with confirmation	Must-Have	High
F-006-RQ-002	Execute withdrawals	User withdrawal requests processed with approvals	Must-Have	High
F-006-RQ-003	Balance reconciliation	Real-time balance updates with external system sync	Must-Have	Medium
F-006-RQ-004	Transaction batching	Multiple transactions batched for gas optimization	Should-Have	Medium
F-006-RQ-005	Multi-signature support	Critical transactions require multiple approvals	Must-Have	High
F-006-RQ-006	Fee calculation	Accurate fee calculation for all transaction types	Must-Have	Medium

Technical Specifications:

- **Input Parameters**: Transaction type (deposit, withdrawal), amount, currency, user ID, approval requirements
- **Output/Response**: Transaction ID, status, estimated fees, approval requirements, processing timeline
- **Performance Criteria**: Deposit confirmation < 5 minutes, withdrawal processing < 30 minutes, balance update < 10 seconds
- **Data Requirements**: Supported currencies, fee structures, approval thresholds, transaction limits

Validation Rules:

- **Business Rules**: Minimum/maximum transaction limits, daily withdrawal limits, multi-signature thresholds
- **Data Validation**: Positive amounts only, supported currency validation, sufficient balance verification
- **Security Requirements**: Multi-signature wallet integration, transaction signing security, fraud detection
- **Compliance Requirements**: Transaction reporting thresholds, AML screening integration, regulatory limits

2.2.5 Regulatory Reporting Requirements (F-008)

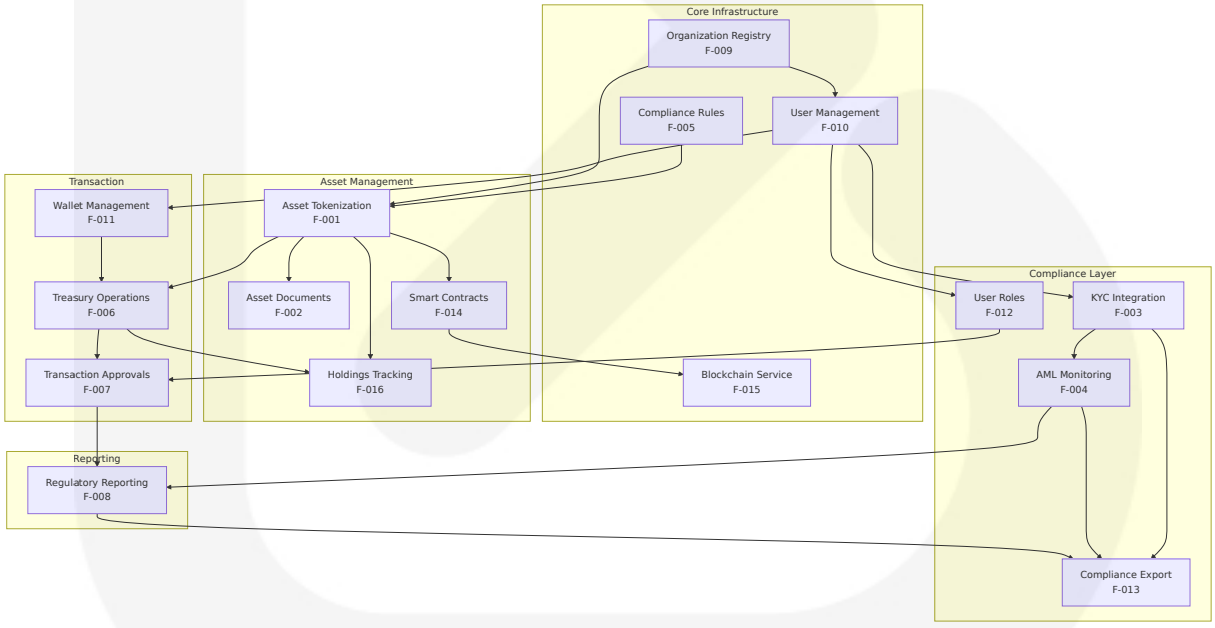
Requirement ID	Description	Acceptance Criteria	Priority	Complexity
F-008-RQ-001	Generate SAR reports	Suspicious Activity Reports generated per regulatory format	Must-Have	High
F-008-RQ-002	Generate CTR reports	Currency Transaction Reports for transactions >$10K	Must-Have	High
F-008-RQ-003	Schedule report generation	Automated report generation based on configurable schedules	Should-Have	Medium
F-008-RQ-004	Multiple export formats	Reports available in PDF, Excel, JSON, XML formats	Should-Have	Low
F-008-RQ-005	Report delivery system	Secure delivery of reports to regulatory authorities	Could-Have	Medium
F-008-RQ-006	Report validation	Generated reports validated against regulatory schemas	Must-Have	Medium

Technical Specifications:

- **Input Parameters**: Report type, date range, organization ID, delivery preferences, format selection
- **Output/Response**: Report ID, generation status, download URL, validation results, delivery confirmation
- **Performance Criteria**: Report generation < 5 minutes for standard reports, < 30 minutes for complex reports, concurrent generation support
- **Data Requirements**: Report templates, regulatory schemas, data aggregation rules, formatting specifications

2.3 FEATURE RELATIONSHIPS

2.3.1 Feature Dependencies Diagram



2.3.2 Integration Points Matrix

Feature A	Feature B	Integration Type	Data Flow	Interface
KYC Integration (F-003)	User Management (F-010)	Direct API	User verification status	REST API
Compliance Rules (F-005)	Asset Tokenization (F-001)	Event-driven	Rule evaluation results	Event Bus
Compliance Rules (F-005)	Treasury Operations (F-006)	Synchronous	Transaction validation	REST API
Asset Tokenization (F-001)	Smart Contracts (F-014)	Service call	Contract deployment	gRPC
Audit Logging (F-013)	All Features	Event streaming	Audit events	Message Queue
Treasury Operations (F-006)	Blockchain Service (F-015)	Direct integration	Transaction broadcast	WebSocket
AML Monitoring (F-004)	Regulatory Reporting (F-008)	Data pipeline	Suspicious activity data	Batch API

2.3.3 Shared Components and Services

Authentication Middleware (auth-middleware):

- Used by: All protected endpoints across all services
- Purpose: JWT validation, rate limiting, request logging
- Dependencies: Redis for session storage, PostgreSQL for user data

Database Package (packages/database):

- Used by: All services requiring data persistence
- Purpose: Shared Prisma client, model definitions, migrations
- Dependencies: PostgreSQL, connection pooling

Compliance Middleware (packages/compliance_middleware):

- Used by: All transaction and user management services
- Purpose: Real-time compliance checking, rule evaluation
- Dependencies: Compliance Rules Engine (F-005), Redis for caching

Audit Logger:

- Used by: All services for compliance tracking
- Purpose: Dual-write audit logging, event correlation
- Dependencies: Message queue, audit database

2.4 IMPLEMENTATION CONSIDERATIONS

2.4.1 Asset Tokenization (F-001) Implementation

Technical Constraints:

- Gas cost optimization for smart contract deployment across multiple chains
- Multi-chain deployment complexity and chain-specific considerations
- Custody provider API rate limits and integration complexities

Performance Requirements:

- Smart contract deployment completion within 60 seconds
- Asset metadata operations sub-second response time
- Support for concurrent tokenization requests

Scalability Considerations:

- Horizontal scaling for API endpoints with stateless design
- Single-writer pattern for blockchain operations to prevent conflicts

- Caching strategy for frequently accessed asset metadata

Security Implications:

- Private key management for contract deployment with HSM integration
- Multi-signature approval workflow for high-value asset deployments
- Secure storage and transmission of custody provider credentials

Maintenance Requirements:

- Smart contract upgrade patterns and migration strategies
- Regular security audits for contract code changes
- Monitoring and alerting for deployment failures

2.4.2 KYC Integration (F-003) Implementation

Technical Constraints:

- KYC provider API rate limits requiring intelligent load balancing
- Data residency requirements across different jurisdictions
- Provider-specific data format and response handling

Performance Requirements:

- 30-second SLA for verification request initiation
- Sub-5-second webhook processing with high availability
- Result caching to reduce provider API calls

Scalability Considerations:

- Provider failover mechanisms for high availability
- Result caching strategy with Redis for performance
- Concurrent verification processing with queue management

Security Implications:

- PII handling with encryption at rest and in transit
- Document storage with hash verification and integrity checking
- Access logging and audit trail for all document operations

Maintenance Requirements:

- Provider API version updates and compatibility management
- Credential rotation and security key management
- Monitoring for provider availability and response times

2.4.3 Compliance Rules Engine (F-005) Implementation

Technical Constraints:

- Rule complexity limitations to prevent performance degradation
- YAML parsing and validation overhead
- Circular dependency detection in rule definitions

Performance Requirements:

- Rule evaluation completion under 100ms (p95), 200ms (p99)
- Support for concurrent rule evaluations exceeding 1000 TPS
- Real-time rule updates without system downtime

Scalability Considerations:

- Rule caching with intelligent cache invalidation
- Horizontal scaling with stateless rule evaluation
- Load balancing across multiple evaluation engines

Security Implications:

- Rule tampering detection and prevention mechanisms
- Approval workflow for rule changes with audit trail
- Encrypted storage of sensitive compliance rules

Maintenance Requirements:

- Rule versioning system with rollback capabilities
- Regular rule validation and testing procedures
- Performance monitoring and optimization

2.4.4 Treasury Operations (F-006) Implementation

Technical Constraints:

- Multi-chain transaction complexity and gas optimization
- Payment processor integration limitations and fees
- Multi-signature wallet coordination challenges

Performance Requirements:

- Deposit confirmation within 5 minutes for supported currencies
- Withdrawal processing completion within 30 minutes
- Real-time balance updates with sub-10-second propagation

Scalability Considerations:

- Transaction batching for gas cost optimization
- Asynchronous processing for non-blocking operations
- Load balancing for high-volume transaction periods

Security Implications:

- Multi-signature wallet integration for large transactions
- Transaction signing security with hardware wallets
- Fraud detection and prevention mechanisms

Maintenance Requirements:

- Regular reconciliation with external systems
- Monitoring for failed transactions and manual intervention

- Gas price optimization strategies

2.4.5 Regulatory Reporting (F-008) Implementation

Technical Constraints:

- Report generation complexity and size limitations
- Regulatory format requirements and validation schemas
- Data aggregation across multiple services and timeframes

Performance Requirements:

- Standard report generation within 5 minutes
- Complex report generation within 30 minutes
- Concurrent report generation support for multiple users

Scalability Considerations:

- Asynchronous report generation with queue-based processing
- Distributed computing for large dataset processing
- Caching of frequently requested report data

Security Implications:

- Data access control and user permission validation
- Secure report delivery and download mechanisms
- Audit trail for report access and generation

Maintenance Requirements:

- Template updates for regulatory requirement changes
- Performance optimization for large dataset reports
- Regular validation against regulatory schemas

2.5 NON-FUNCTIONAL REQUIREMENTS

2.5.1 Performance Requirements

Requirement Type	Specification	Target Metric	Measurement Method
API Response Time	95th percentile response time	< 200ms	APM monitoring (Datadog/NewRelic)
Database Query Performance	Complex queries	< 500ms	Query performance monitoring
System Uptime	Service availability	99.99% SLA	Uptime monitoring
Concurrent Users	Simultaneous active users	10,000+ users	Load testing with k6
Transaction Throughput	Peak transaction processing	10,000+ TPS	Performance benchmarking
Report Generation	Standard compliance reports	< 5 minutes	Automated testing

2.5.2 Security Requirements

Security Domain	Requirements	Implementation
Authentication	JWT-based with refresh tokens	Auth middleware package
Authorization	Role-based access control (RBAC)	User roles and permissions (F-012)
Data Encryption	AES-256 at rest, TLS 1.3 in transit	Database and API layer
Audit Logging	Comprehensive audit trail	Dual-write audit logging pattern
PII Protection	Data masking and encryption	KYC service implementation
Multi-Factor Authentication	WebAuthn support	Identity service integration

2.5.3 Scalability Requirements

Component	Scaling Strategy	Target Capacity
API Gateway	Horizontal auto-scaling	1000+ req/sec per instance
Database	Read replicas + connection pooling	10,000+ concurrent connections
KYC Processing	Queue-based with worker pools	100+ concurrent verifications
Report Generation	Async processing with job queues	50+ concurrent report generations
Blockchain Operations	Single-writer with optimistic locking	Chain-specific optimization

2.6 TRACEABILITY MATRIX

Feature ID	Business Requirement	API Endpoints	Database Tables	Services	Test Coverage
F-001	Asset Onboarding Flow	POST /api/assets/tokenize GET /api/assets/:id	products, product_documents, custody_configs	gateway, blockchain-service, policy-service	Unit: 80%, Integration: 70%
F-003	Investor Management	POST /api/compliance/kyc GET /api/compliance/status/:userId	users, compliance_verifications, kyc_results	kyc-provider, identity-service	Unit: 85%, Integration: 75%
F-005	Compliance Rules	POST /api/policies/evaluate	compliance_rules, rule_evaluations	policy-service, compliance-service	Unit: 90%, Integration: 80%

Feature ID	Business Requirement	API Endpoints	Database Tables	Services	Test Coverage
		PUT /api/policies/:id			
F-006	Treasury Operations	POST /api/treasury/deposit POST /api/treasury/withdraw	transactions, holdings, treasury_balances	gateway, blockchain-service	Unit: 75%, Integration: 65%
F-008	Regulatory Reporting	POST /reports/sar POST /reports/ctr	regulatory_reports, report_data, audit_logs	regulatory-reporting, audit-log-writer	Unit: 70%, Integration: 60%
F-013	Compliance Export	GET /audit/export/:period POST /audit/generate	audit_logs, export_requests, evidence_files	audit-log-writer, file-storage	Unit: 80%, Integration: 70%

References

Files Examined:

- `packages/database/models.py` - Core data model definitions and entity relationships for all features
- `docs/API.md` - High-level API endpoint documentation and authentication specifications
- `docs/PRODUCT_REQUIREMENTS.md` - Comprehensive PRD v3.0 with detailed business requirements
- `PRD.md` - Master PRD index with module references and business context

- `README.md` - Project overview, technology stack, and development setup
- `docs/IMPLEMENTATION_STATUS.md` - Current implementation progress tracking across all components
- `docs/ROADMAP.md` - 15-week phased delivery schedule and milestone tracking

Folders Explored:

- `services/` - 11 microservices with detailed capability analysis and implementation status
- `packages/` - Shared packages including auth-middleware, database, and SDK components
- `packages/database/` - Database models, migrations, and relationship definitions
- `apps/` - Frontend applications including compliance dashboard and user interfaces
- `docs/` - Comprehensive documentation suite with architecture and business requirements
- `contracts/` - Smart contract infrastructure with ERC-3643 compliance implementation
- `services/kyc-provider/` - KYC integration service with multi-provider orchestration
- `services/regulatory-reporting/` - Reporting service with SAR/CTR generation capabilities

Technical Specification Sections:

- `1.1 EXECUTIVE SUMMARY` - Business context, stakeholders, and value proposition
- `1.2 SYSTEM OVERVIEW` - Architecture description, technical approach, and success criteria
- `1.3 SCOPE` - In-scope elements, implementation boundaries, and exclusion criteria

3. TECHNOLOGY STACK

3.1 PROGRAMMING LANGUAGES

3.1.1 Primary Language Selection

TypeScript v5.3.3 - v5.4.5 serves as the primary programming language across the entire Veria platform, chosen for its type safety benefits in a financial compliance environment where data integrity is critical. All 11 microservices (gateway, identity-service, policy-service, compliance-service, audit-log-writer, blockchain-service, kyc-provider, regulatory-reporting, tool-masker, ai-broker, graph-service) implement TypeScript with strict mode enabled and ES2020/ES2022 compilation targets.

JavaScript (Node.js v20) provides the runtime environment, selected for its mature ecosystem, extensive library support for financial integrations, and proven scalability in enterprise applications. The platform leverages ES modules ("type": "module") across all services to ensure modern JavaScript standards and optimal tree-shaking capabilities.

3.1.2 Secondary Language Components

Python 3.11+ handles specialized blockchain integration and compliance processing through two dedicated packages:

- **Blockchain Integration Package** (`packages/blockchain`): Implements Web3.py for Ethereum/Polygon interactions, AsyncIO for real-time event monitoring, and SQLAlchemy ORM patterns for blockchain state management
- **Compliance Middleware** (`packages/compliance_middleware`): Uses FastAPI for high-performance API endpoints and Pydantic for rigorous data validation in compliance rule processing

Solidity 0.8.20 implements ERC-3643 compliant smart contracts including VeriaSecurityToken, ModularCompliance, and IdentityRegistry contracts. This version provides critical security enhancements and gas optimization features essential for enterprise tokenization.

3.1.3 Language Selection Justification

The multi-language approach addresses specific domain requirements:

- **TypeScript**: Ensures type safety for financial data processing and provides superior developer experience for large codebases
- **Python**: Leverages mature blockchain and data science ecosystems for compliance analytics
- **Solidity**: Industry standard for security token contracts with extensive auditing tools

3.2 FRAMEWORKS & LIBRARIES

3.2.1 Backend Framework Architecture

Fastify v4.25.2 - v4.28.1 serves as the primary web framework across all microservices, chosen for its exceptional performance characteristics (up to 30,000 requests per second) and built-in schema validation. This selection supports the platform's requirement for API response times under 200ms (p99) and 10,000+ TPS throughput.

Core Fastify ecosystem integration:

- **@fastify/cors v8.5.0**: Cross-origin resource sharing with configurable security policies
- **@fastify/helmet v11.1.1**: Security header management for compliance with financial industry security standards
- **@fastify/jwt v8.0.1**: JWT token handling with automatic verification and role-based access control

Express maintains limited usage in the ai-broker service, representing a legacy pattern being phased out in favor of the standardized Fastify architecture.

3.2.2 Frontend Framework Stack

React v18 provides the foundational UI framework, selected for its component-based architecture that supports the platform's complex form-heavy interfaces for asset onboarding and investor management. The declarative programming model aligns with compliance workflows requiring clear state management and audit trails.

Next.js v14 implements the main frontend application using the App Router architecture, enabling:

- Server-side rendering for improved initial load performance
- Static site generation for compliance documentation pages
- Built-in API route handling for frontend-specific endpoints
- Environment variable injection for multi-environment deployments

Vite powers the compliance-dashboard build system, providing:

- Hot Module Replacement (HMR) for rapid development cycles
- Optimized production builds with code splitting
- Proxy configuration for seamless API integration during development

3.2.3 Validation & Security Libraries

Zod v3.22.4 - v3.25.76 implements runtime schema validation across all services, ensuring data integrity for financial transactions and compliance data. This library choice supports the platform's zero-tolerance policy for data validation errors in regulatory environments.

Pino v8.17.2 with **pino-pretty v10.3.1** provides structured logging throughout the platform, enabling comprehensive audit trails required for

regulatory compliance and system monitoring.

3.2.4 UI Component Framework

Tailwind CSS implements utility-first styling, chosen for its consistency across large development teams and rapid prototype capabilities essential for compliance interface development.

Radix UI provides headless, accessible components that ensure WCAG compliance for institutional users, particularly important for enterprise clients with accessibility requirements.

Supporting UI libraries:

- **Recharts**: Data visualization for compliance reporting and analytics dashboards
- **Lucide React**: Consistent iconography throughout the application
- **clsx + tailwind-merge**: Dynamic className management for conditional styling

3.3 OPEN SOURCE DEPENDENCIES

3.3.1 Core Runtime Dependencies

Database & Caching Clients:

- **pg v8.11.3 - v8.16.3**: PostgreSQL client with connection pooling for high-performance database operations
- **ioredis v5.3.2 - v5.7.0**: Redis client supporting advanced features like clustering and sentinel modes for session management and caching

Security & Authentication:

- **bcrypt v5.1.1**: Password hashing with configurable salt rounds for user credential security
- **jsonwebtoken**: JWT generation and verification supporting RS256 algorithm for enhanced security
- **@simplewebauthn/server v13.1.2**: WebAuthn implementation for passwordless authentication, supporting FIDO2 security keys

HTTP & Network:

- **axios**: HTTP client with interceptor support for API request/response transformation
- **node-fetch v3.3.2**: Fetch API implementation for Node.js, maintaining consistency with browser-based fetch usage

3.3.2 Blockchain Integration Dependencies

Ethereum Ecosystem:

- **ethers v6.9.0**: Comprehensive Ethereum library providing wallet management, contract interaction, and transaction signing capabilities
- **Web3.py**: Python-based Ethereum integration supporting multiple provider endpoints and event filtering

Smart Contract Development:

- **Hardhat v2.19.4**: Ethereum development environment providing compilation, testing, and deployment tools
- **@openzeppelin/contracts v5.0.1**: Battle-tested smart contract libraries for ERC-3643 token implementation
- **@openzeppelin/contracts-upgradeable v5.0.1**: Upgradeable contract patterns for iterative compliance rule deployment
- **TypeChain v8.3.2**: TypeScript bindings generation for type-safe smart contract interactions

3.3.3 Document Generation & Processing

Financial Document Generation:

- **pdfkit v0.14.0**: PDF generation for compliance reports and investor documentation
- **exceljs v4.4.0**: Excel file manipulation for regulatory reporting formats
- **handlebars v4.7.8**: Template engine for dynamic report generation
- **node-cron v3.0.3**: Scheduled task execution for automated compliance reporting

3.3.4 Development & Testing Framework

Package Management:

- **pnpm v10**: Workspace-aware package manager providing efficient dependency management for monorepo architecture

Testing Ecosystem:

- **vitest v1.1.0 - v1.6.0**: Fast unit testing framework with native TypeScript support and coverage reporting
- **Playwright**: End-to-end testing for compliance workflow validation
- **k6**: Performance testing supporting the 10,000+ TPS scalability requirement

Build & Compilation:

- **tsx v4.6.2 - v4.7.0**: TypeScript execution engine for development workflows
- **esbuild**: High-performance JavaScript bundler for optimized production builds

3.4 THIRD-PARTY SERVICES

3.4.1 Payment Processing Integration

Stripe v14.25.0 provides payment processing capabilities with PCI DSS compliance, essential for handling investor fund flows. The integration supports:

- Tokenized payment methods for recurring compliance fees
- Webhook processing for real-time payment status updates
- Multi-currency support for international investors
- Comprehensive fraud detection and prevention

Configuration requires `STRIPE_SECRET_KEY` and `STRIPE_WEBHOOK_SECRET` environment variables.

3.4.2 KYC/AML Provider Ecosystem

The platform implements a multi-provider architecture for Know Your Customer (KYC) and Anti-Money Laundering (AML) compliance, currently configured with mock implementations for development:

Blockchain Analytics:

- **Chainalysis** (`CHAINALYSIS_API_KEY`): Blockchain transaction monitoring and risk scoring
- **TRM Labs** (`TRM_API_KEY`): Cryptocurrency compliance and investigation tools

Identity Verification:

- **Jumio** (`JUMIO_API_TOKEN`): Document verification and biometric identity matching
- **Onfido** (`ONFIDO_API_TOKEN`): Identity verification with liveness detection

This multi-provider approach ensures redundancy and coverage across different verification requirements while maintaining the flexibility to adapt to evolving compliance regulations.

3.4.3 Blockchain Infrastructure Services

Node Providers:

- **Alchemy** (`ALCHEMY_API_KEY`): Primary blockchain node provider offering enhanced APIs, webhooks, and analytics
- **Supported Networks**:
  - Polygon Network (Chain ID: 137) - Primary production network
  - Mumbai Testnet (Chain ID: 80001) - Polygon testing environment
  - Ethereum Mainnet (Chain ID: 1) - Alternative deployment target
  - Sepolia Testnet (Chain ID: 11155111) - Ethereum testing environment

3.4.4 Monitoring & Analytics Services

Application Performance Monitoring:

- **Sentry** (`SENTRY_DSN`): Real-time error tracking and performance monitoring with alerting capabilities
- **Datadog** (`DATADOG_API_KEY`): Infrastructure monitoring, APM, and log aggregation
- **Prometheus/Grafana**: Metrics collection and visualization for system observability

These services support the platform's 99.99% uptime SLA requirement and provide critical insights for maintaining performance targets.

3.4.5 Cloud Infrastructure Services

Google Cloud Platform (Primary deployment platform):

- **Cloud Run**: Serverless container hosting with automatic scaling
- **Artifact Registry**: Docker image storage and management
- **Secret Manager**: Secure credential and API key management

Amazon Web Services (Alternative deployment option):

- **EKS**: Managed Kubernetes service for container orchestration

- **ECR**: Elastic Container Registry for Docker image storage
- **Secrets Manager**: Secure secrets storage and rotation

3.5 DATABASES & STORAGE

3.5.1 Primary Database Architecture

PostgreSQL v14 serves as the primary relational database, selected for its robust ACID compliance, advanced indexing capabilities, and excellent performance characteristics required for financial data integrity. The database configuration implements:

- **Connection Pooling**: QueuePool with pool_size=20 and max_overflow=10 for optimal connection management
- **Database Schemas**:
  - Development: `veria_dev`
  - Production: `veria`
- **Integration Approach**: Direct pg client usage for complex queries with minimal Prisma ORM implementation

PostgreSQL's selection supports the platform's requirements for:

- Complex relational queries across asset, investor, and compliance data
- Transaction isolation for financial operations
- Advanced JSON column support for flexible compliance rule storage
- Excellent backup and recovery capabilities for regulatory data retention

3.5.2 Caching & Session Management

Redis v7 implements distributed caching and session management with specific configurations for compliance platform requirements:

Caching Strategies:

- **Session Management**: 7-day TTL for user sessions with sliding expiration
- **Rate Limiting**: 60-second windows for API endpoint protection
- **Policy Caching**: 300-second TTL for compliance rule evaluation results
- **KYC Result Caching**: 1-hour TTL for verification status to reduce provider API calls
- **Token Blacklisting**: 24-hour TTL for revoked JWT tokens

Redis cluster configuration supports high availability and automatic failover, critical for maintaining system uptime during peak compliance processing periods.

3.5.3 Vector Database for Semantic Search

Qdrant provides vector similarity search capabilities, supporting semantic search across compliance documentation and regulatory text. The configuration includes:

- **HTTP Port**: 6333 for REST API access
- **gRPC Port**: 6334 for high-performance operations
- **Use Cases**: Document similarity matching, compliance rule suggestion, and regulatory text analysis

3.5.4 File Storage & Audit Systems

File Storage Implementation:

- **Audit Logs**: Local filesystem storage (`.audit-data/audit.log`) with structured logging format
- **Temporary Files**: System-managed temporary storage for report generation
- **Document Generation**: In-memory PDF and Excel creation with secure cleanup

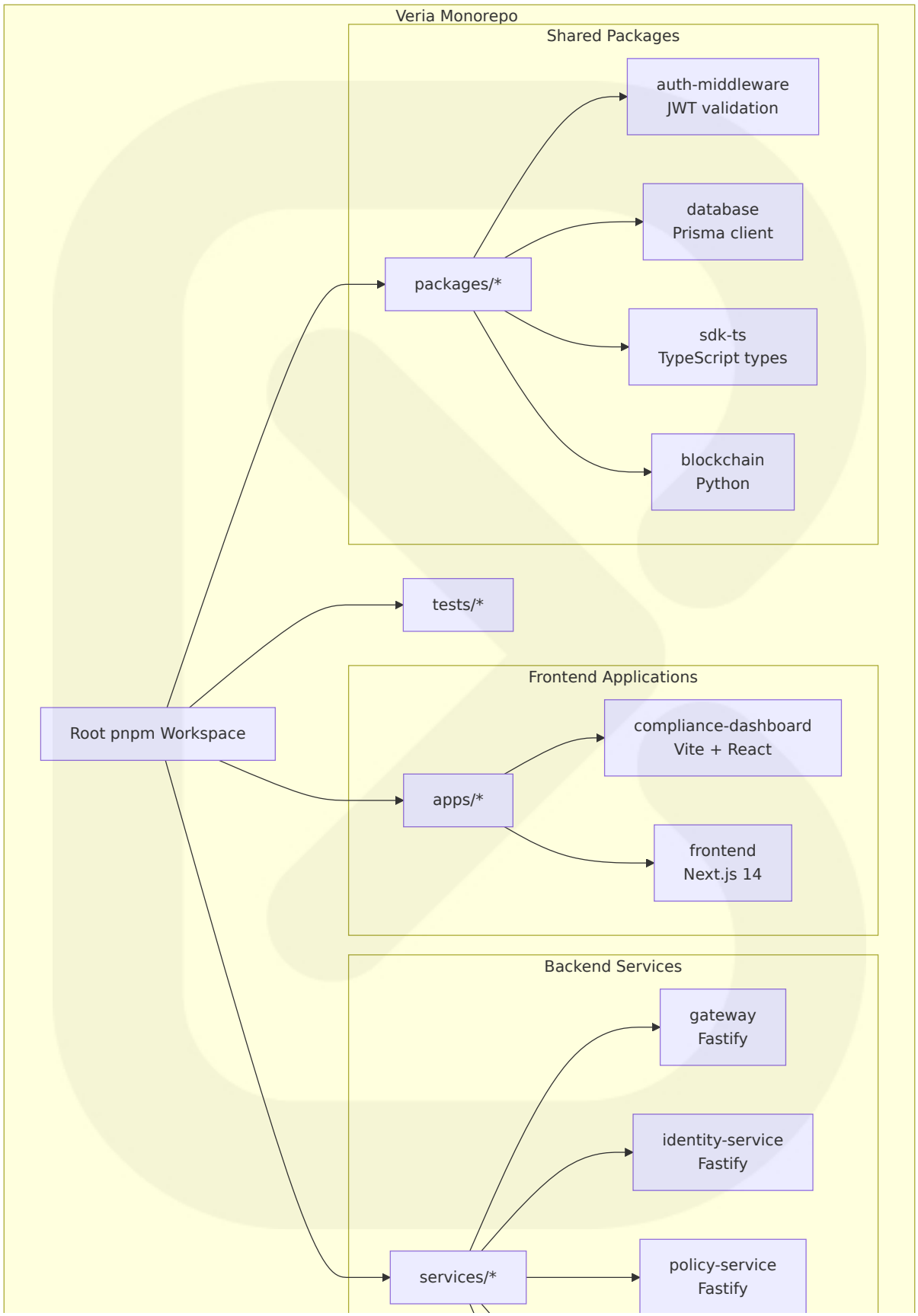
Data Persistence Strategy:

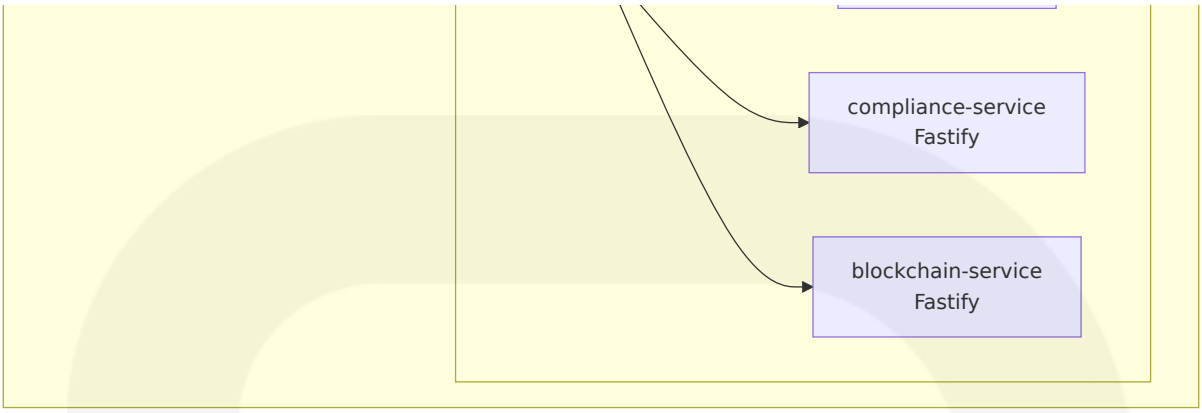
- **Transactional Data**: PostgreSQL with daily backups and point-in-time recovery
- **Cache Data**: Redis with persistence enabled for session continuity
- **Audit Trails**: Immutable log storage with hash verification for compliance auditing

3.6 DEVELOPMENT & DEPLOYMENT

3.6.1 Development Environment Architecture

Monorepo Management with pnpm Workspaces:
The platform utilizes pnpm workspaces for efficient monorepo management across multiple application types:


Veria Monorepo
Shared Packages
auth-middleware
JWT validation
database
Prisma client
packages/*
sdk-ts
TypeScript types
blockchain
Python
tests/*
Frontend Applications
compliance-dashboard
Vite + React
Root pnpm Workspace
apps/*
frontend
Next.js 14
Backend Services
gateway
Fastify
identity-service
Fastify
services/*
policy-service
Fastify



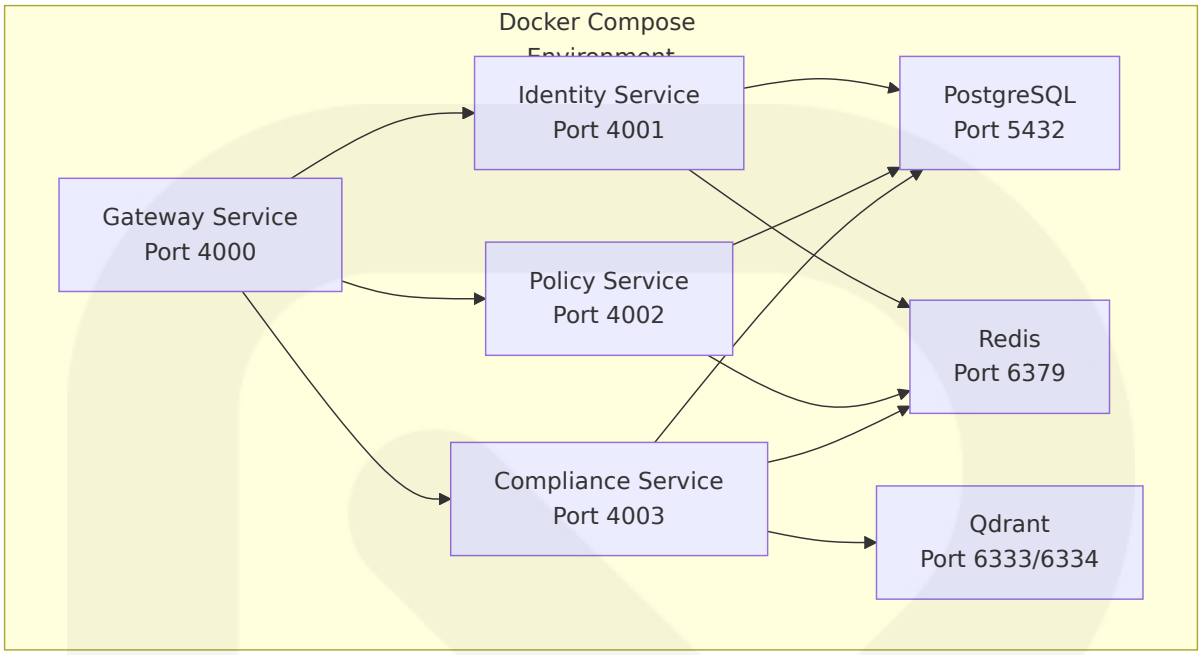
Build Dependency Management:
The workspace configuration requires specific build ordering due to internal package dependencies. Shared packages must be built before dependent services, managed through pnpm's workspace dependency resolution.

3.6.2 Containerization Strategy

Docker Implementation:

- **Base Image**: Node.js 20-alpine for minimal attack surface and reduced image size
- **Multi-stage Builds**: Separate build and runtime stages for optimized production images
- **Security Configuration**: Non-root user execution (UID 1001) for enhanced container security
- **Development Orchestration**: Docker Compose v3.8 with service dependencies and health checks

Container Architecture:



3.6.3 Build Systems & Compilation

Frontend Build Tools:

- **Vite**: Compliance dashboard with HMR support and proxy configuration for API integration
- **Next.js**: Main frontend application with App Router, SSR capabilities, and environment variable injection
- **TypeScript Compiler (tsc)**: Service compilation with strict type checking and ES2020 target

Smart Contract Build Pipeline:

- **Hardhat**: Contract compilation, testing, and deployment with TypeScript integration
- **Solidity Compiler**: Version 0.8.20 with optimization enabled for gas efficiency
- **TypeChain**: Automatic TypeScript binding generation for type-safe contract interactions

3.6.4 CI/CD Pipeline Architecture

GitHub Actions Implementation:

- **Matrix Builds**: Parallel builds across multiple services for reduced pipeline duration
- **Docker Integration**: Automated image building and pushing to multiple registries
- **Quality Gates**: Automated testing, linting, and security scanning before deployment

Container Registry Strategy:

- **Primary**: GitHub Container Registry (ghcr.io) for integrated GitHub Actions workflows
- **Cloud-Specific**: Google Artifact Registry and AWS ECR for platform-specific deployments
- **Multi-Registry Push**: Ensures availability across different deployment environments

3.6.5 Deployment Platforms & Infrastructure

Primary Deployment: Google Cloud Run:

- **Serverless Containers**: Automatic scaling based on request volume
- **Managed Infrastructure**: No server management required, focusing development on business logic
- **Environment Configuration**: Knative Service manifests for declarative deployments

Alternative Deployment: AWS EKS:

- **Kubernetes Orchestration**: Helm Charts (./infra/helm/veria) for complex deployment scenarios
- **Container Management**: Full control over container orchestration and scaling policies

- **Multi-Zone Deployment**: High availability across multiple availability zones

Infrastructure as Code:

- **Terraform**: Infrastructure provisioning and management (referenced in user context)
- **Helm Charts**: Kubernetes application deployment and configuration management
- **Staged Deployments**: Progressive rollouts with automated rollback capabilities

3.6.6 Testing & Quality Assurance

Testing Framework Ecosystem:

- **Vitest**: Unit and integration testing with coverage reporting and mock support
- **Playwright**: End-to-end testing for critical compliance workflows
- **k6**: Performance testing validating 10,000+ TPS requirements
- **Hardhat Test**: Smart contract testing with mainnet forking capabilities
- **pytest**: Python package testing for blockchain and compliance middleware

Code Quality Tools:

- **ESLint**: JavaScript/TypeScript linting with financial industry best practices
- **Prettier**: Consistent code formatting across the entire codebase
- **Husky**: Git hooks for pre-commit quality checks
- **Trivy**: Security vulnerability scanning for containers and dependencies
- **pnpm audit**: Dependency vulnerability assessment and remediation guidance

References

Files Examined

- `package.json` - Root workspace configuration and shared dependencies
- `docker-compose.yml` - Infrastructure services configuration
- `services/gateway/package.json` - Gateway service dependencies
- `contracts/package.json` - Smart contract development dependencies
- `services/identity-service/package.json` - Identity service framework stack
- `packages/database/package.json` - Database client configuration
- `services/blockchain-service/package.json` - Blockchain integration libraries
- `.env.example` - External service configurations
- `Dockerfile` - Container runtime environment
- `cloudrun.yaml` - Cloud deployment configuration
- `pnpm-workspace.yaml` - Monorepo workspace structure
- `services/compliance-service/package.json` - Compliance service dependencies
- `services/regulatory-reporting/package.json` - Document generation libraries

Folders Explored

- `/` - Root repository structure and configuration
- `packages/` - Shared packages for auth, database, and SDK components
- `services/` - Microservices architecture with 11 backend services
- `apps/` - Frontend applications including compliance dashboard and main frontend
- `contracts/` - Smart contract development and deployment configuration
- `.github/` - CI/CD workflow configuration and automation

4. PROCESS FLOWCHART

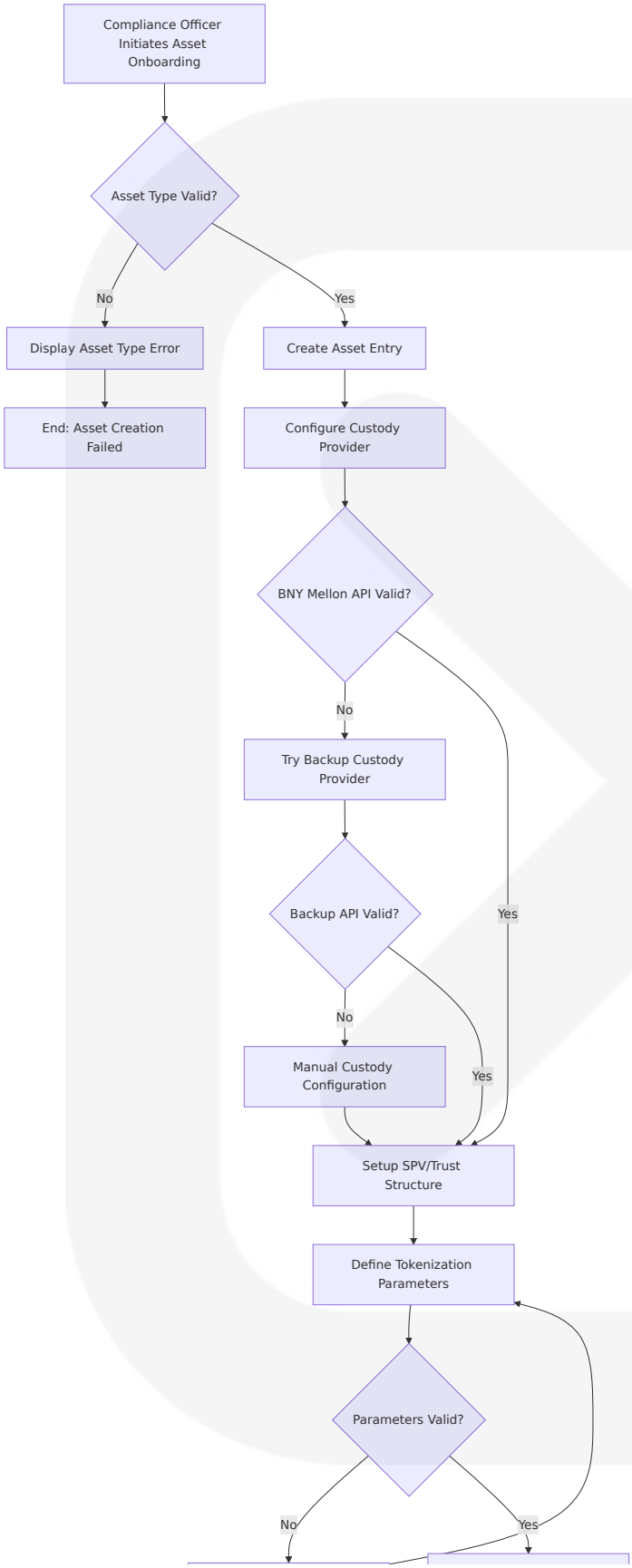
4.1 SYSTEM WORKFLOWS

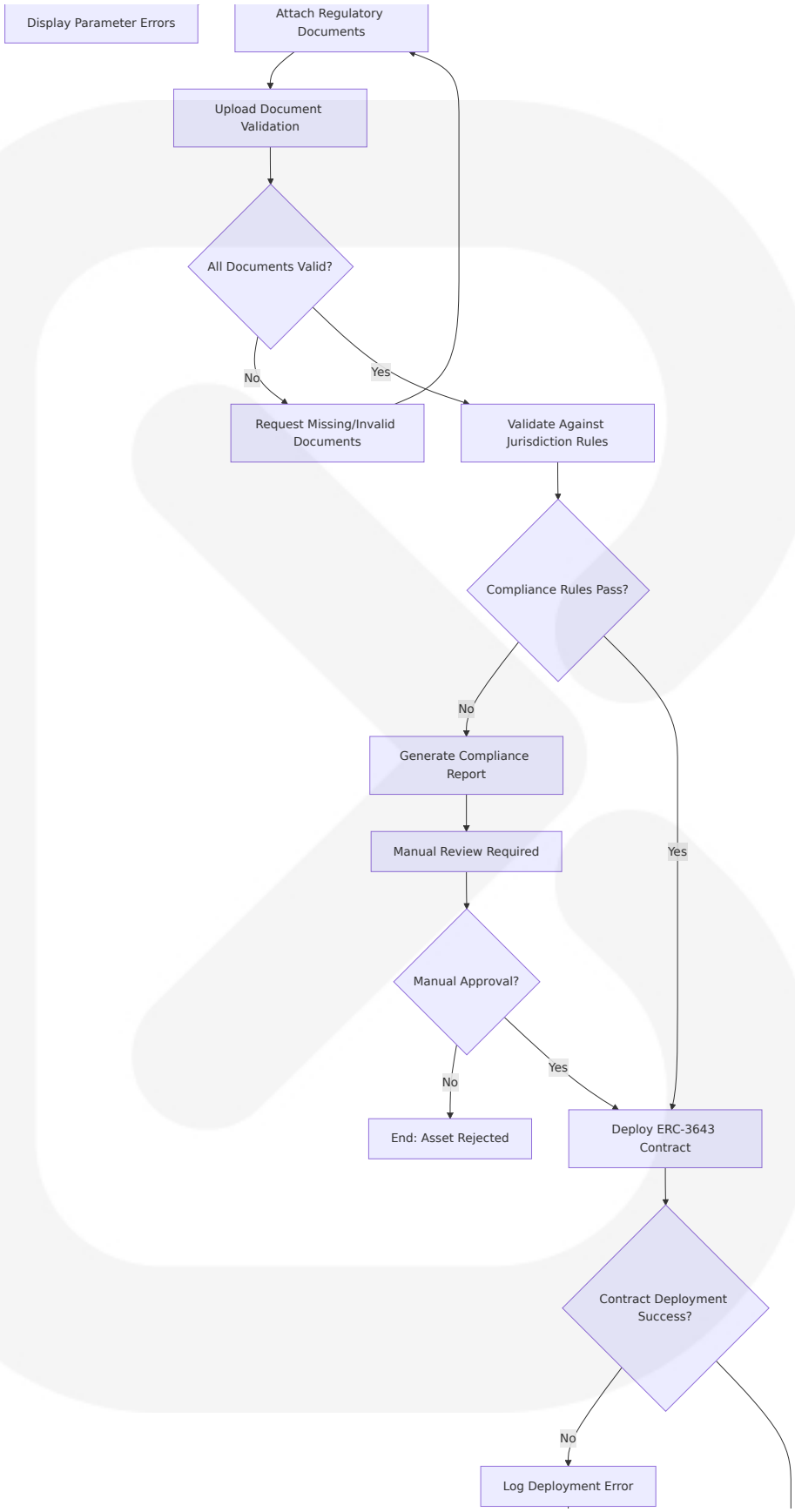
4.1.1 Core Business Processes

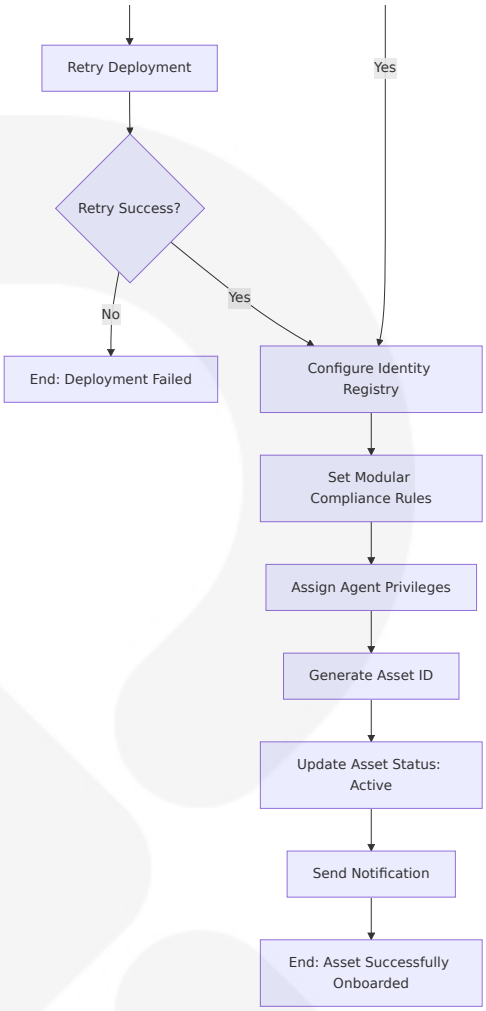
The Veria platform implements six primary business workflows that handle the complete lifecycle of tokenized asset management, from initial asset onboarding through regulatory compliance and reporting. Each workflow incorporates comprehensive validation, error handling, and audit trail generation to ensure regulatory compliance and operational integrity.

4.1.1.1 Asset Onboarding Workflow

The Asset Onboarding Workflow represents the complete lifecycle for tokenizing real-world assets, specifically US Treasuries and Money Market Funds. This workflow integrates with custody providers, implements ERC-3643 compliant smart contracts, and ensures jurisdiction-specific compliance from initiation to deployment.


Compliance Officer Initiates Asset Onboarding
Asset Type Valid?
No
Yes
Display Asset Type Error
End: Asset Creation Failed
Create Asset Entry
Configure Custody Provider
BNY Mellon API Valid?
No
Yes
Try Backup Custody Provider
Backup API Valid?
No
Yes
Manual Custody Configuration
Yes
Setup SPV/Trust Structure
Define Tokenization Parameters
Parameters Valid?
No
Yes


Display Parameter Errors
Attach Regulatory Documents
Upload Document Validation
All Documents Valid?
No
Yes
Request Missing/Invalid Documents
Validate Against Jurisdiction Rules
Compliance Rules Pass?
No
Yes
Generate Compliance Report
Manual Review Required
Manual Approval?
No
Yes
End: Asset Rejected
Deploy ERC-3643 Contract
Contract Deployment Success?
No
Log Deployment Error



Key Decision Points:

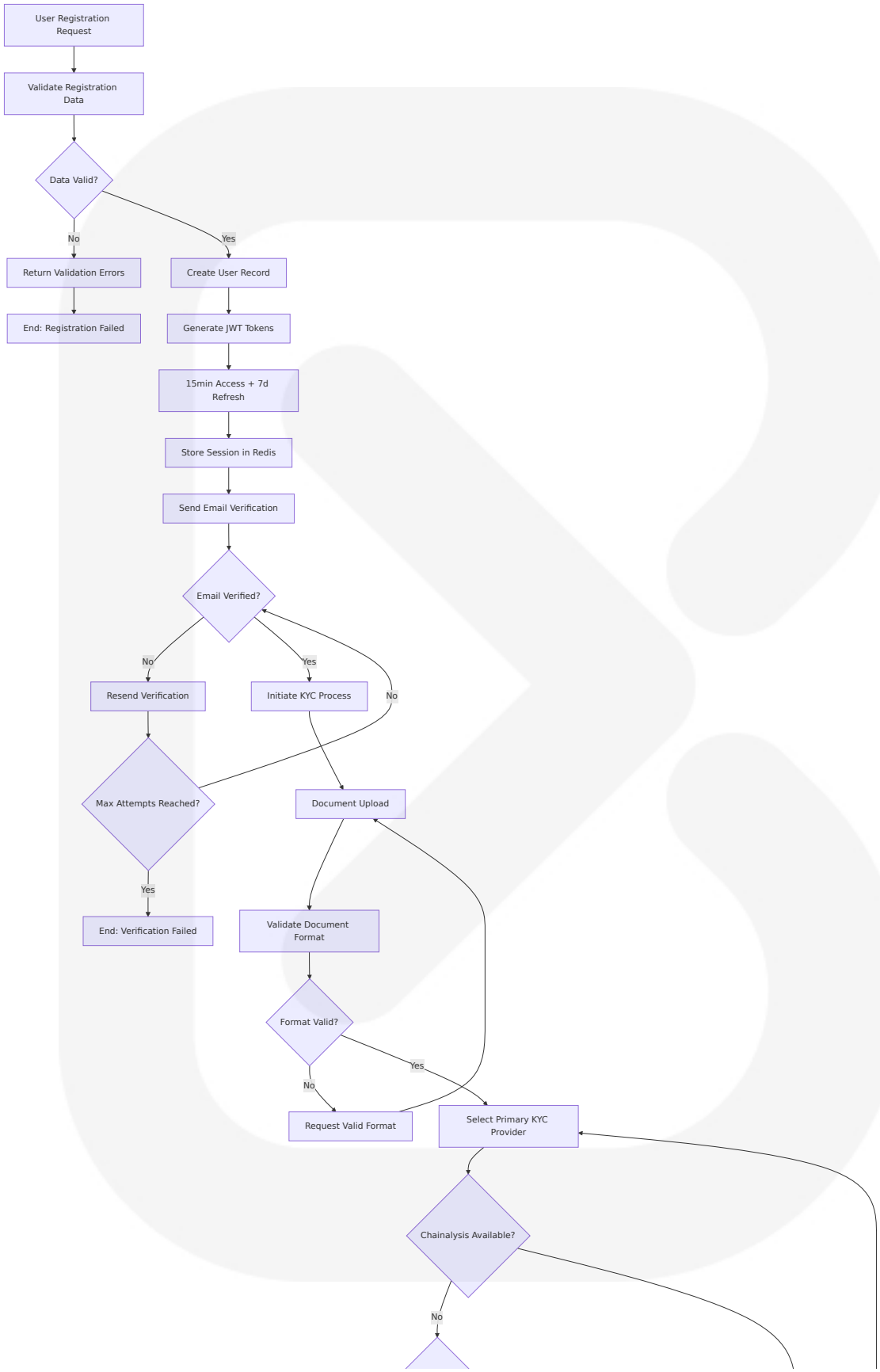
- Asset type validation against supported categories (US Treasuries, Money Market Funds)
- Custody provider API connectivity verification with automatic fallback
- Tokenization parameter validation (symbol format, supply limits, decimal precision)
- Document completeness verification with hash integrity checking
- Jurisdiction-specific compliance rule validation with manual review escalation
- Smart contract deployment success verification with automatic retry logic

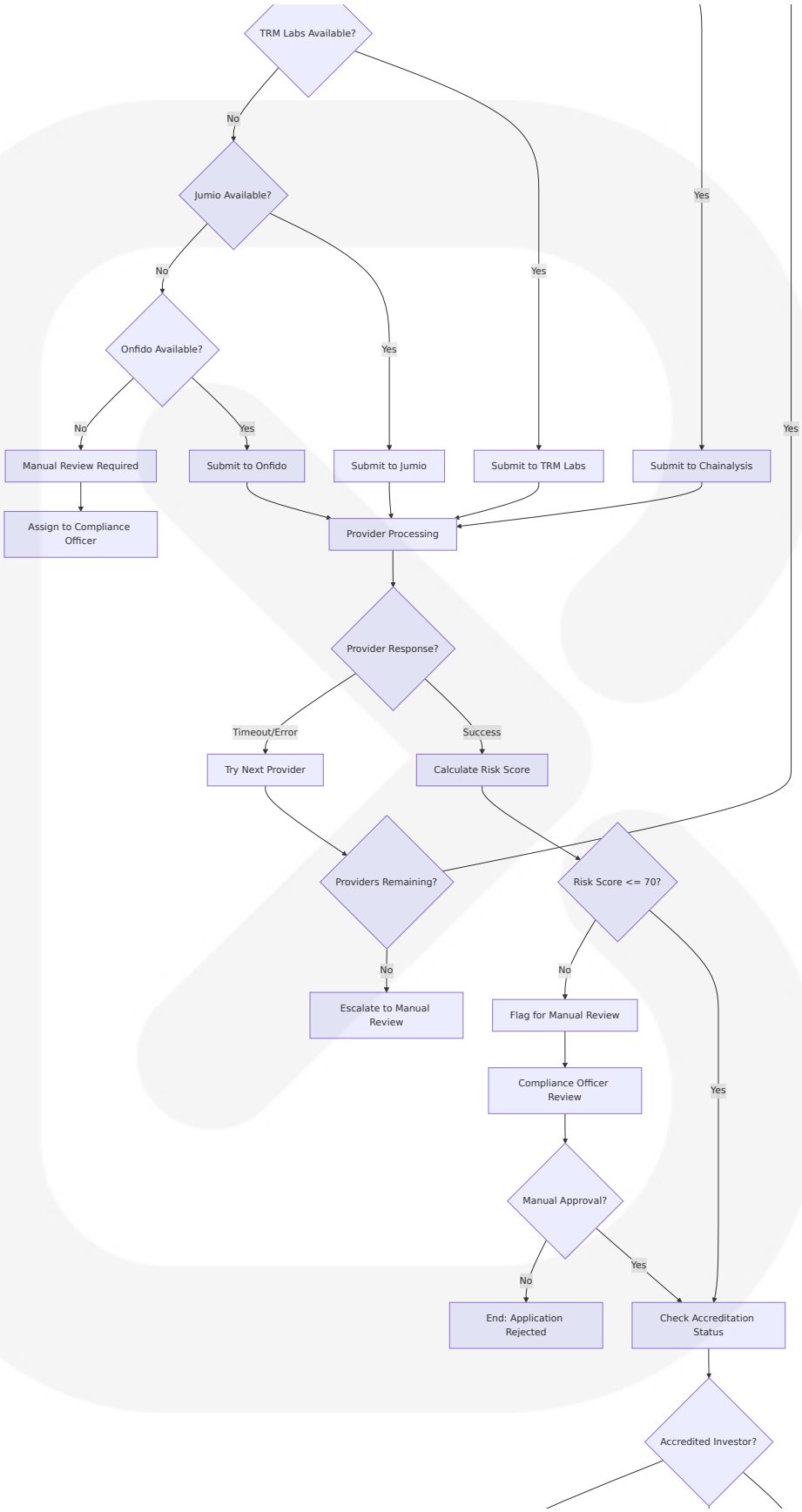
Performance Requirements:

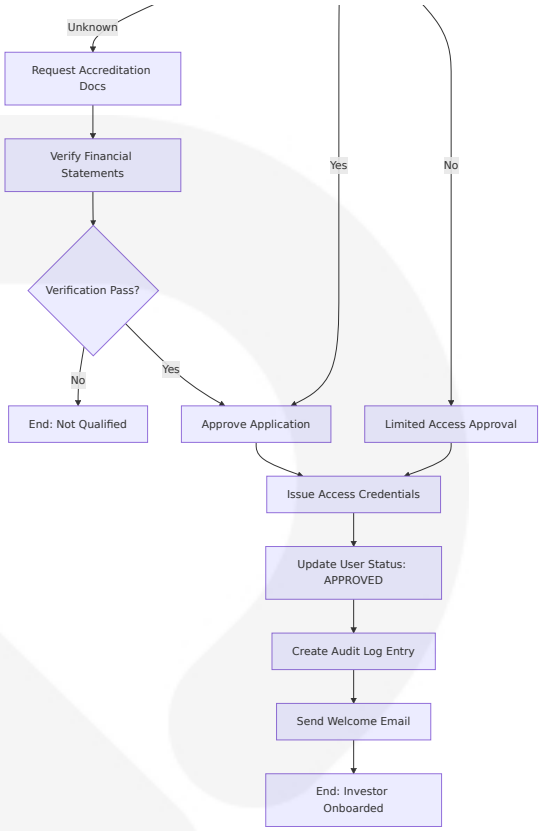
- Contract deployment completion < 60 seconds
- Document validation < 2 seconds per file
- Compliance rule evaluation < 100ms
- End-to-end workflow completion < 5 minutes

4.1.1.2 Investor Management Workflow

The Investor Management Workflow handles the complete investor lifecycle from registration through KYC/AML verification and credential issuance. This workflow implements multi-provider KYC orchestration with fallback capabilities and comprehensive risk assessment.


User Registration Request
Validate Registration Data
Data Valid?
No
Yes
Return Validation Errors
Create User Record
End: Registration Failed
Generate JWT Tokens
15min Access + 7d Refresh
Store Session in Redis
Send Email Verification
Email Verified?
No
Yes
Resend Verification
Initiate KYC Process
No
Max Attempts Reached?
Document Upload
Yes
End: Verification Failed
Validate Document Format
Format Valid?
Yes
No
Request Valid Format
Select Primary KYC Provider
Chainalysis Available?
No


TRM Labs Available?
No
Yes
Jumio Available?
No
Yes
Onfido Available?
No
Yes
Yes
Yes
Yes
Manual Review Required
Submit to Onfido
Submit to Jumio
Submit to TRM Labs
Submit to Chainalysis
Assign to Compliance Officer
Provider Processing
Provider Response?
Timeout/Error
Success
Try Next Provider
Calculate Risk Score
Providers Remaining?
Risk Score <= 70?
No
No
Escalate to Manual Review
Flag for Manual Review
Compliance Officer Review
Yes
Manual Approval?
No
Yes
End: Application Rejected
Check Accreditation Status
Accredited Investor?



Integration Points:

- Multi-provider KYC orchestration with automatic failover
- Redis session management for authentication state
- PostgreSQL user record persistence
- Email verification service integration
- Audit log writer service for compliance tracking

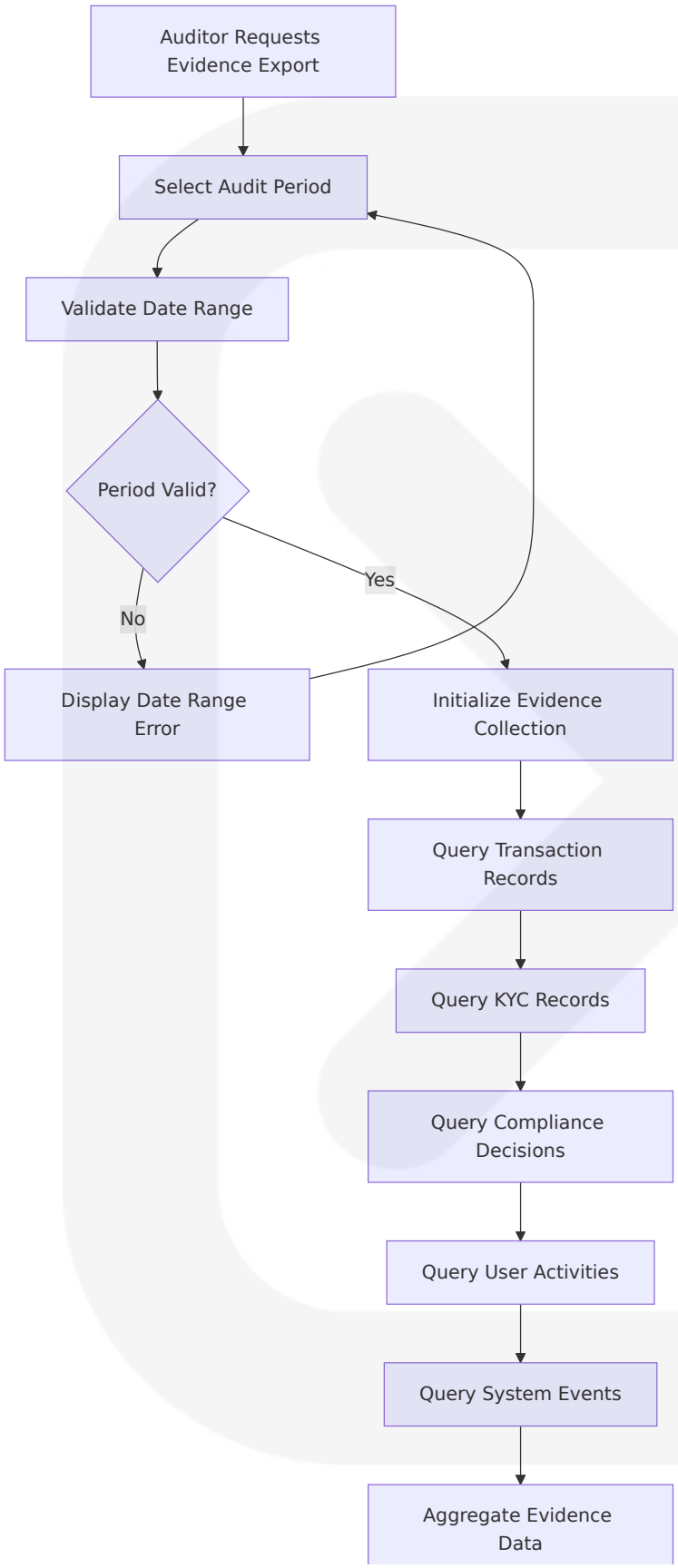
Error Handling:

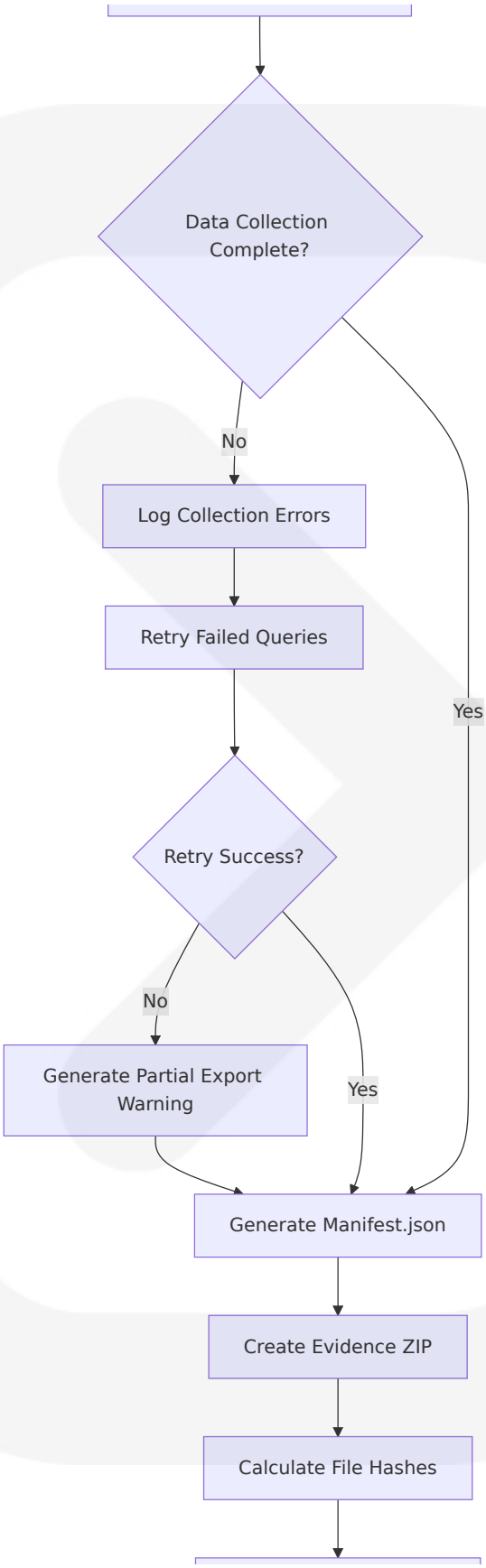
- Provider timeout recovery with automatic provider switching
- Document upload retry mechanism with format validation
- Session expiration handling with refresh token rotation
- Maximum retry limits to prevent infinite loops

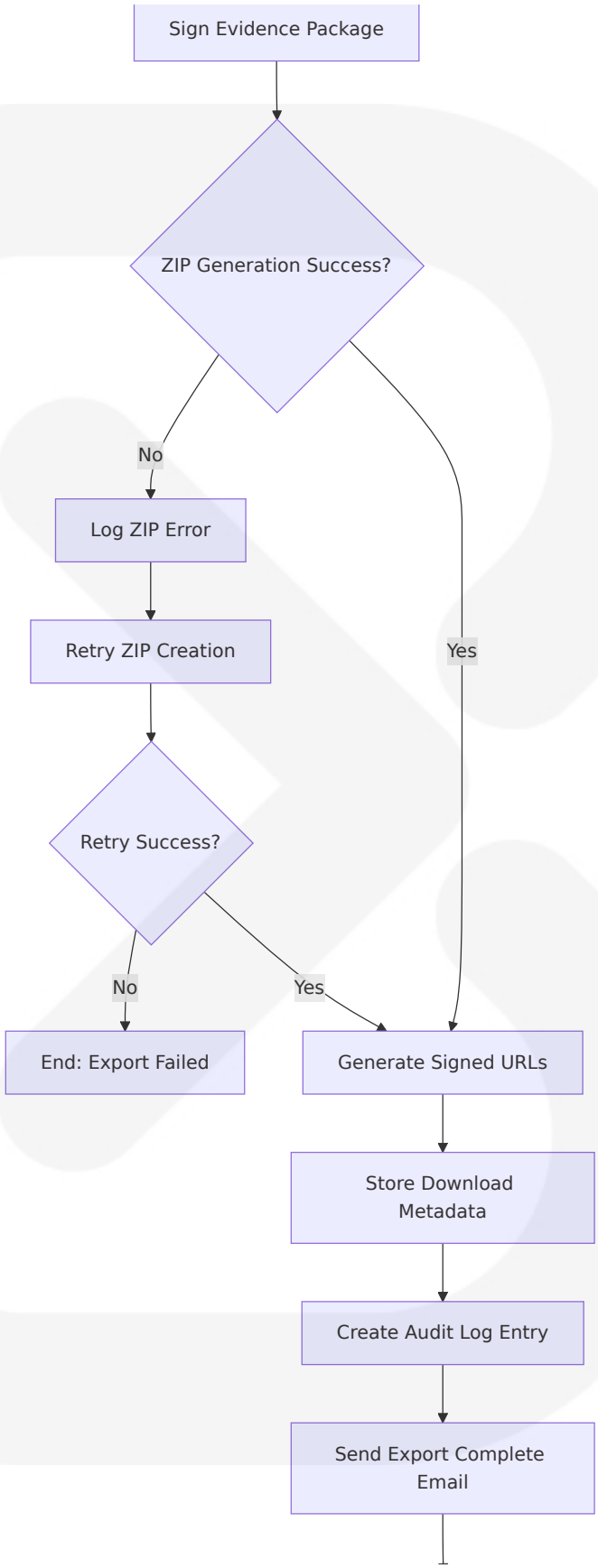
4.1.1.3 Compliance Export Workflow

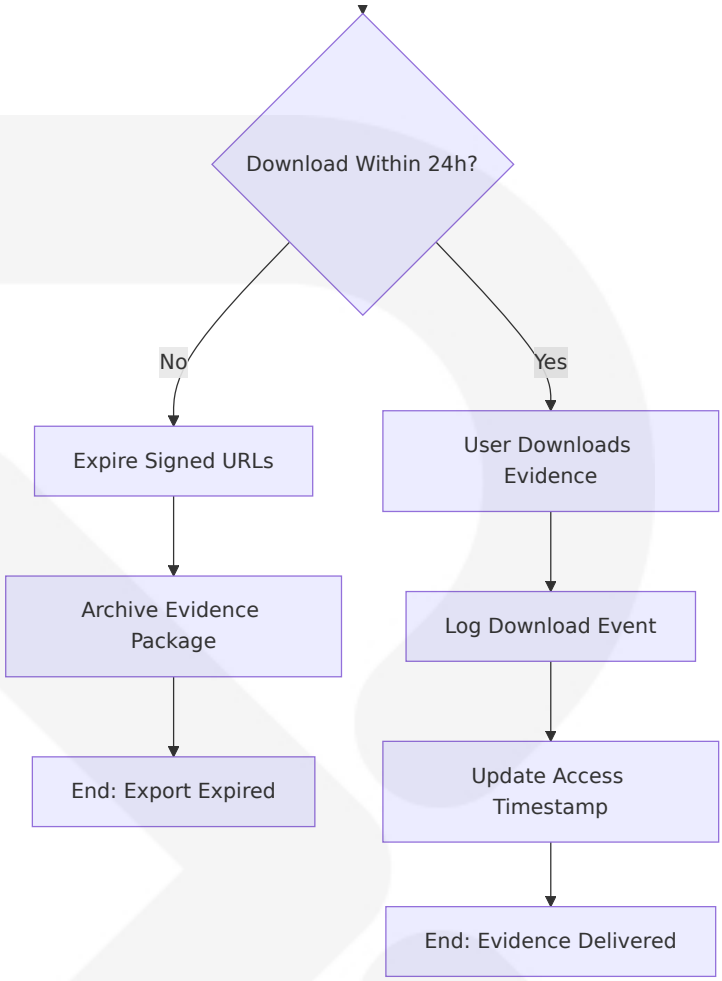
The Compliance Export Workflow enables comprehensive audit evidence collection for regulatory purposes, generating secure, tamper-proof

evidence packages with complete audit trails.


Auditor Requests Evidence Export
Select Audit Period
Validate Date Range
Period Valid?
Yes
No
Display Date Range Error
Initialize Evidence Collection
Query Transaction Records
Query KYC Records
Query Compliance Decisions
Query User Activities
Query System Events
Aggregate Evidence Data


Data Collection Complete?
No
Yes
Log Collection Errors
Retry Failed Queries
Retry Success?
No
Yes
Generate Partial Export Warning
Generate Manifest.json
Create Evidence ZIP
Calculate File Hashes


Sign Evidence Package
ZIP Generation Success?
No
Log ZIP Error
Retry ZIP Creation
Retry Success?
No
End: Export Failed
Yes
Yes
Generate Signed URLs
Store Download Metadata
Create Audit Log Entry
Send Export Complete Email



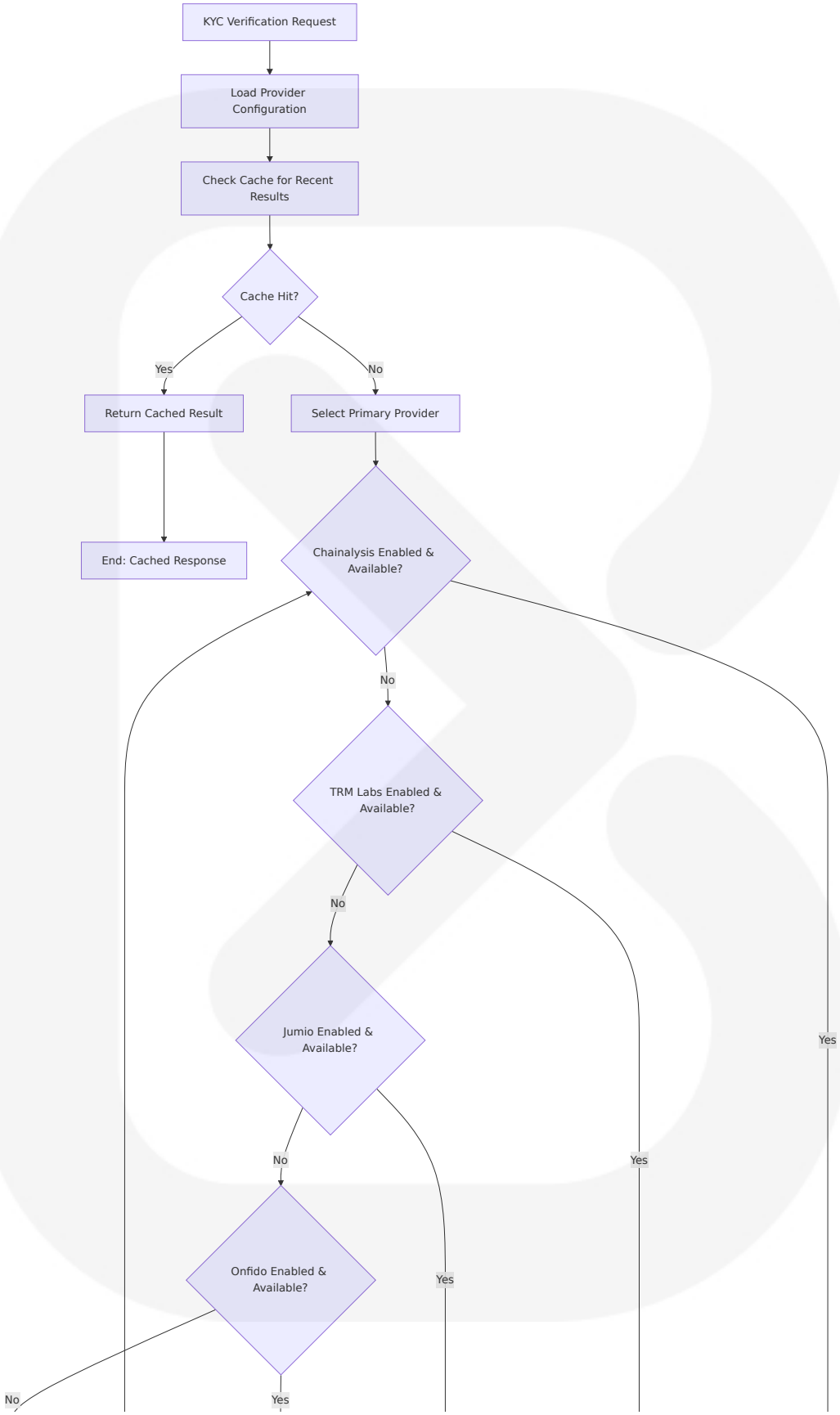
Security Features:

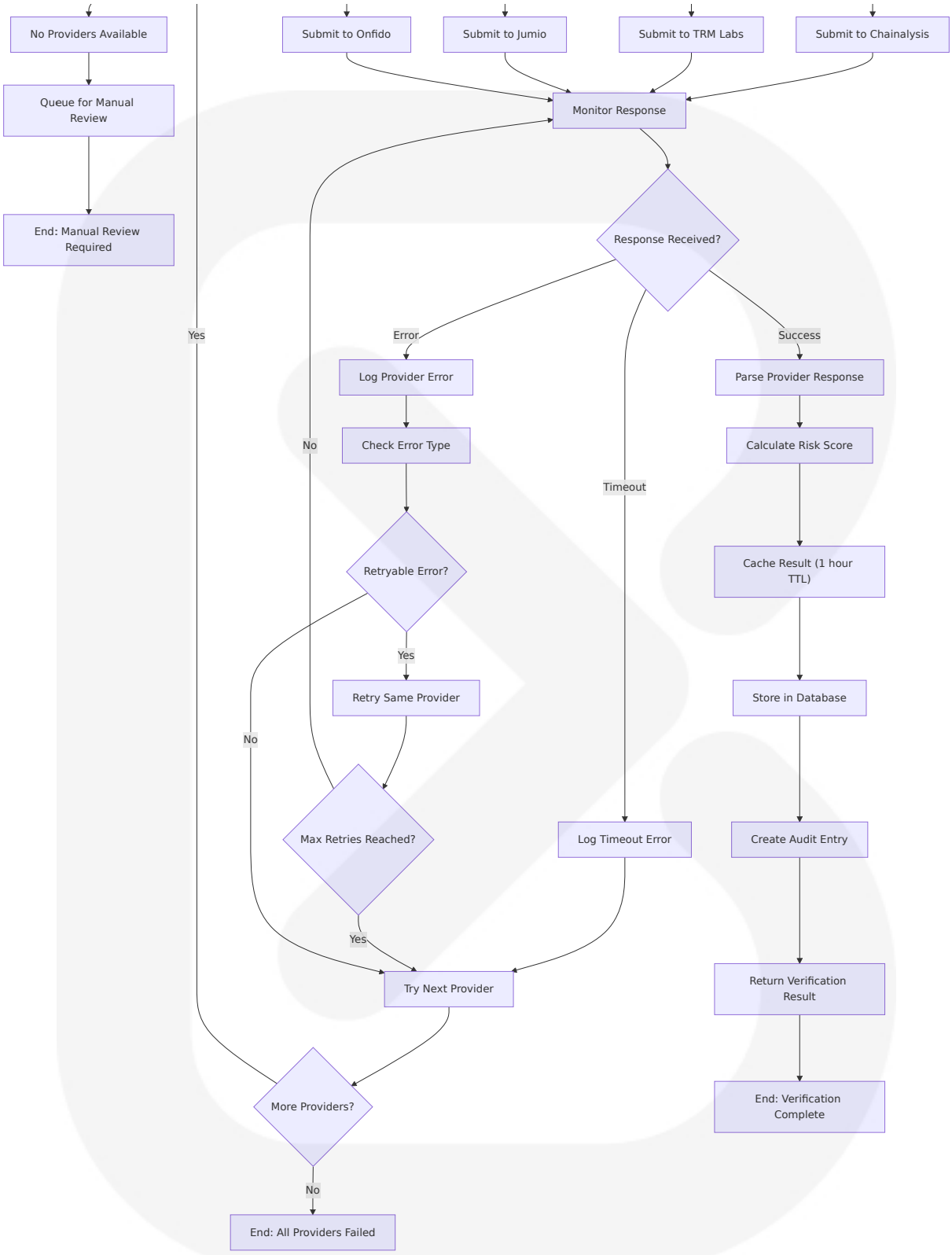
- Evidence package signing for tamper detection
- Signed URL generation with 24-hour expiration
- File hash calculation for integrity verification
- Complete audit trail of export and download activities

4.1.2 Integration Workflows

4.1.2.1 Multi-Provider KYC Orchestration

The system implements sophisticated KYC provider orchestration with intelligent fallback mechanisms and performance optimization through caching.


KYC Verification Request
Load Provider Configuration
Check Cache for Recent Results
Cache Hit?
Yes
No
Return Cached Result
Select Primary Provider
End: Cached Response
Chainalysis Enabled & Available?
No
Yes
TRM Labs Enabled & Available?
No
Yes
Jumio Enabled & Available?
No
Yes
Onfido Enabled & Available?
No
Yes



Provider Configuration:

- Chainalysis: Sanctions screening and wallet analysis

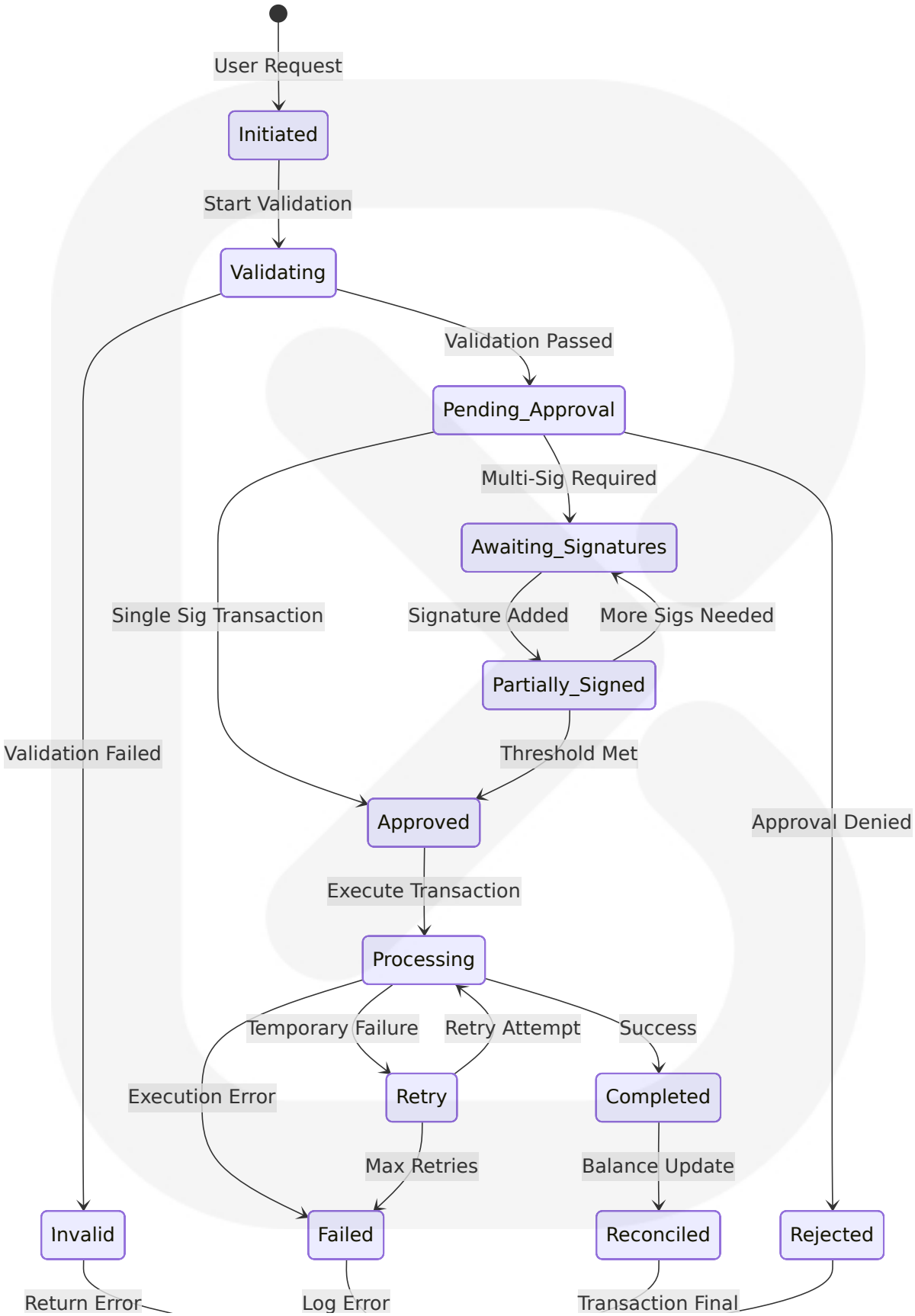
- TRM Labs: Risk scoring and compliance monitoring
- Jumio: Document verification and identity proofing
- Onfido: Biometric verification and document authentication

Performance Optimization:

- Redis caching with 1-hour TTL for repeated verification attempts
- Parallel provider querying where applicable
- Connection pooling for provider API calls
- Response time monitoring and provider ranking

4.1.3 Treasury Operations State Flow

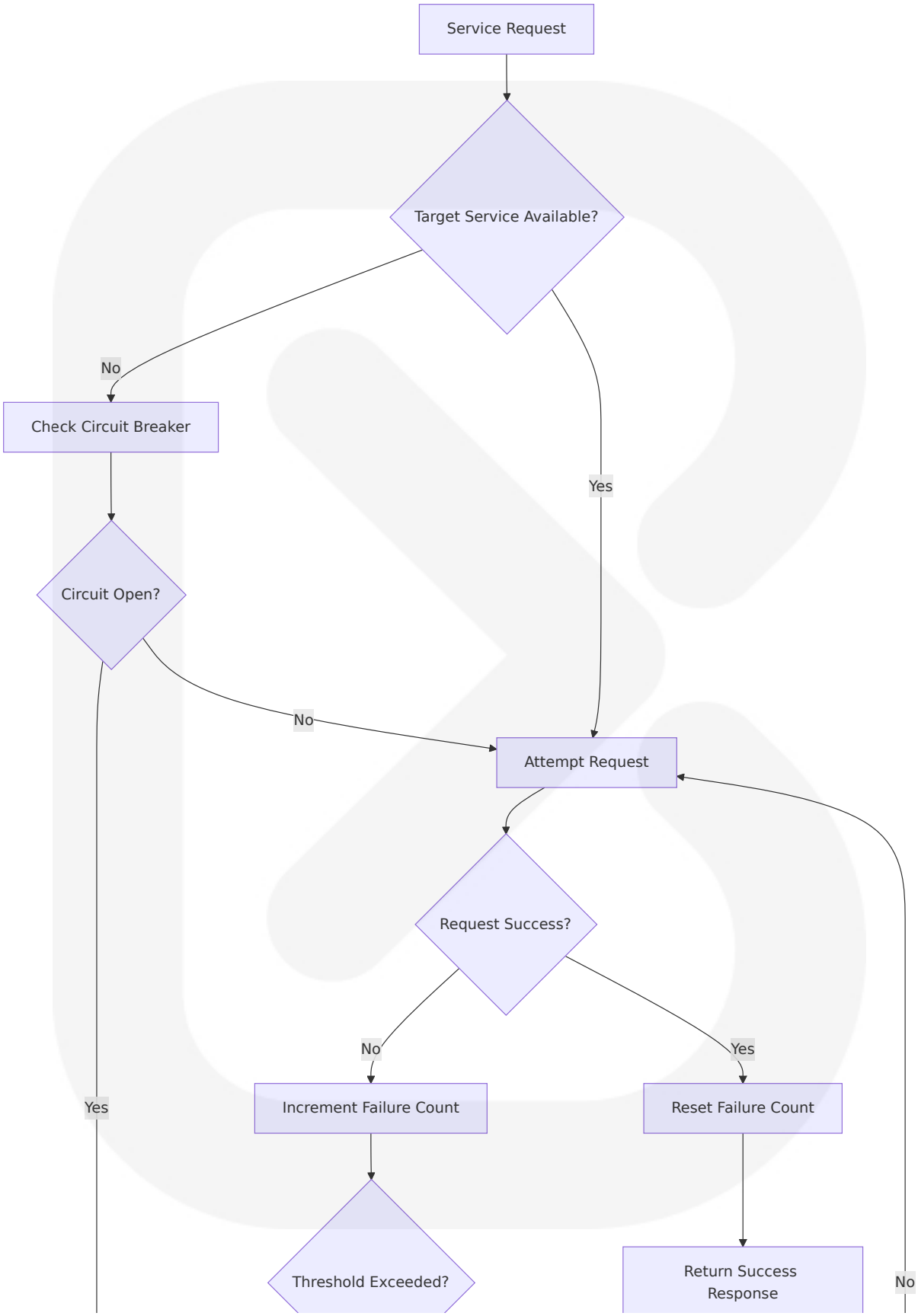
The treasury operations implement a comprehensive state machine for transaction management with multi-signature approval workflows and automated reconciliation.

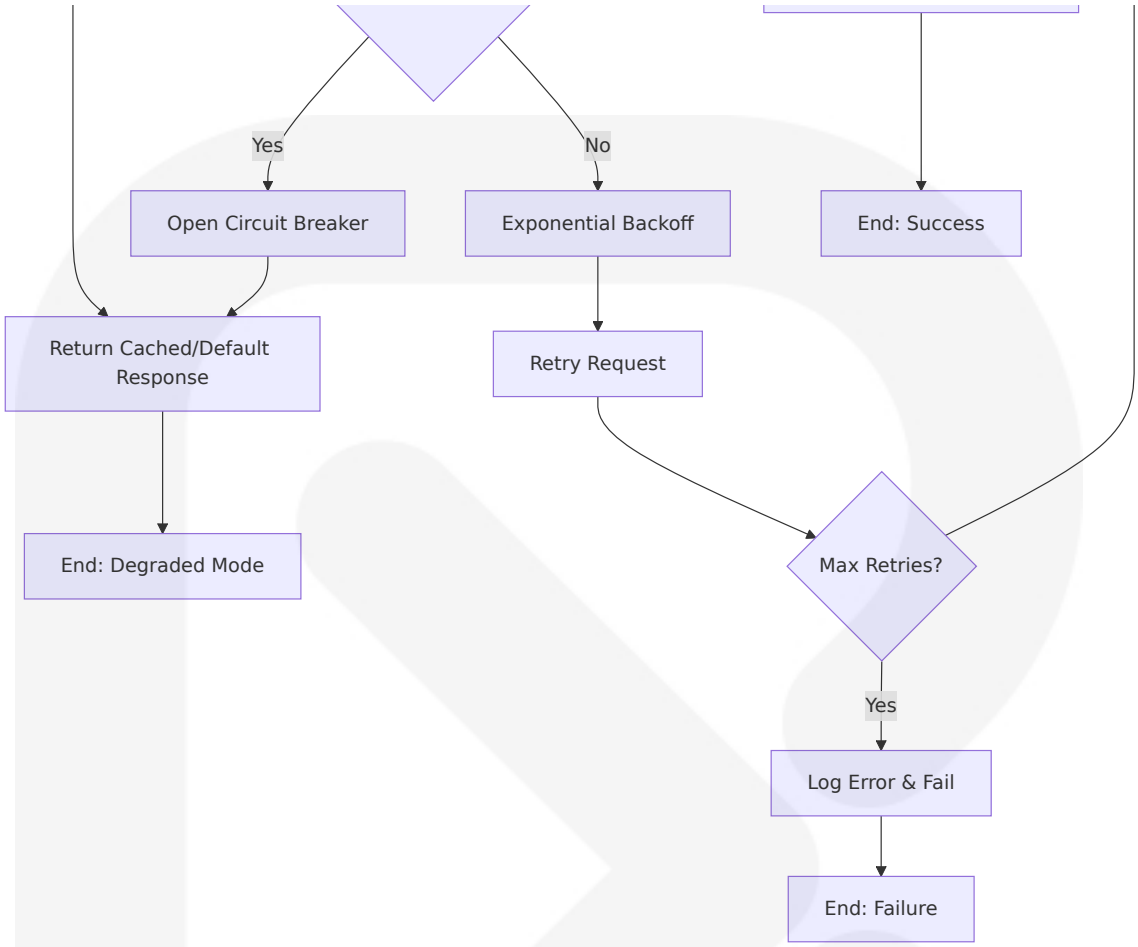

User Request
Initiated
Start Validation
Validating
Validation Passed
Pending_Approval
Multi-Sig Required
Awaiting_Signatures
Single Sig Transaction
Signature Added
More Sigs Needed
Partially_Signed
Validation Failed
Threshold Met
Approved
Approval Denied
Execute Transaction
Processing
Temporary Failure
Retry Attempt
Success
Execution Error
Retry
Completed
Max Retries
Balance Update
Invalid
Failed
Reconciled
Rejected
Return Error
Log Error
Transaction Final

4.2 ERROR HANDLING AND RECOVERY FLOWS

4.2.1 System Resilience Patterns

4.2.1.1 Service Communication Error Handling

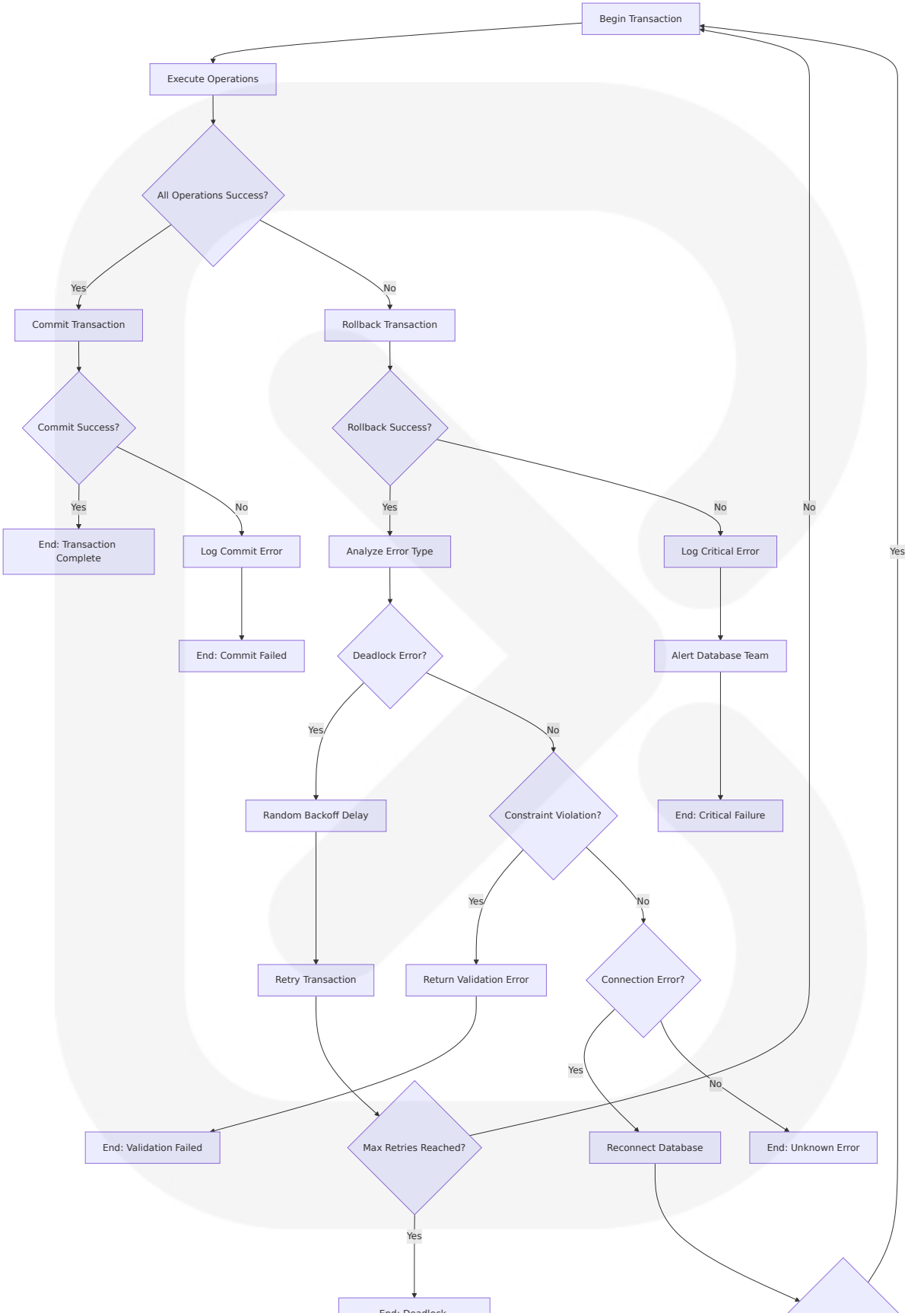

Service Request
Target Service Available?
No
Yes
Check Circuit Breaker
Circuit Open?
No
Yes
Attempt Request
Request Success?
No
Yes
Increment Failure Count
Reset Failure Count
Threshold Exceeded?
Return Success Response
No



Circuit Breaker Configuration:

- Failure threshold: 5 consecutive failures
- Timeout period: 30 seconds
- Half-open retry interval: 10 seconds
- Success threshold for circuit closure: 3 consecutive successes

4.2.1.2 Database Transaction Error Recovery

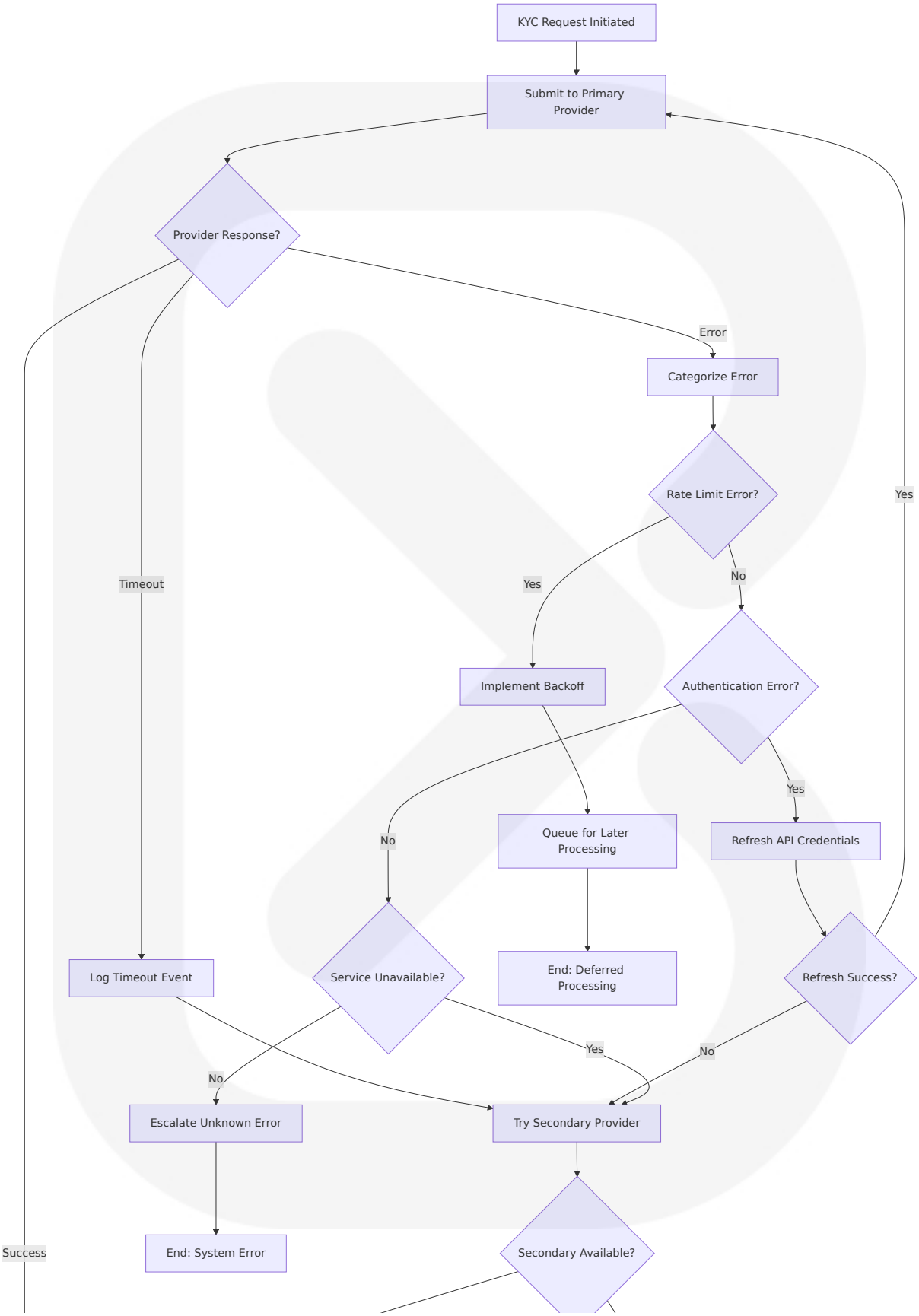

Begin Transaction
Execute Operations
All Operations Success?
Yes
No
Commit Transaction
Rollback Transaction
Commit Success?
Rollback Success?
Yes
No
Yes
No
No
End: Transaction Complete
Log Commit Error
Analyze Error Type
Log Critical Error
Yes
End: Commit Failed
Deadlock Error?
Alert Database Team
Yes
No
Random Backoff Delay
Constraint Violation?
End: Critical Failure
Yes
No
Retry Transaction
Return Validation Error
Connection Error?
Yes
No
End: Validation Failed
Max Retries Reached?
Reconnect Database
End: Unknown Error
Yes

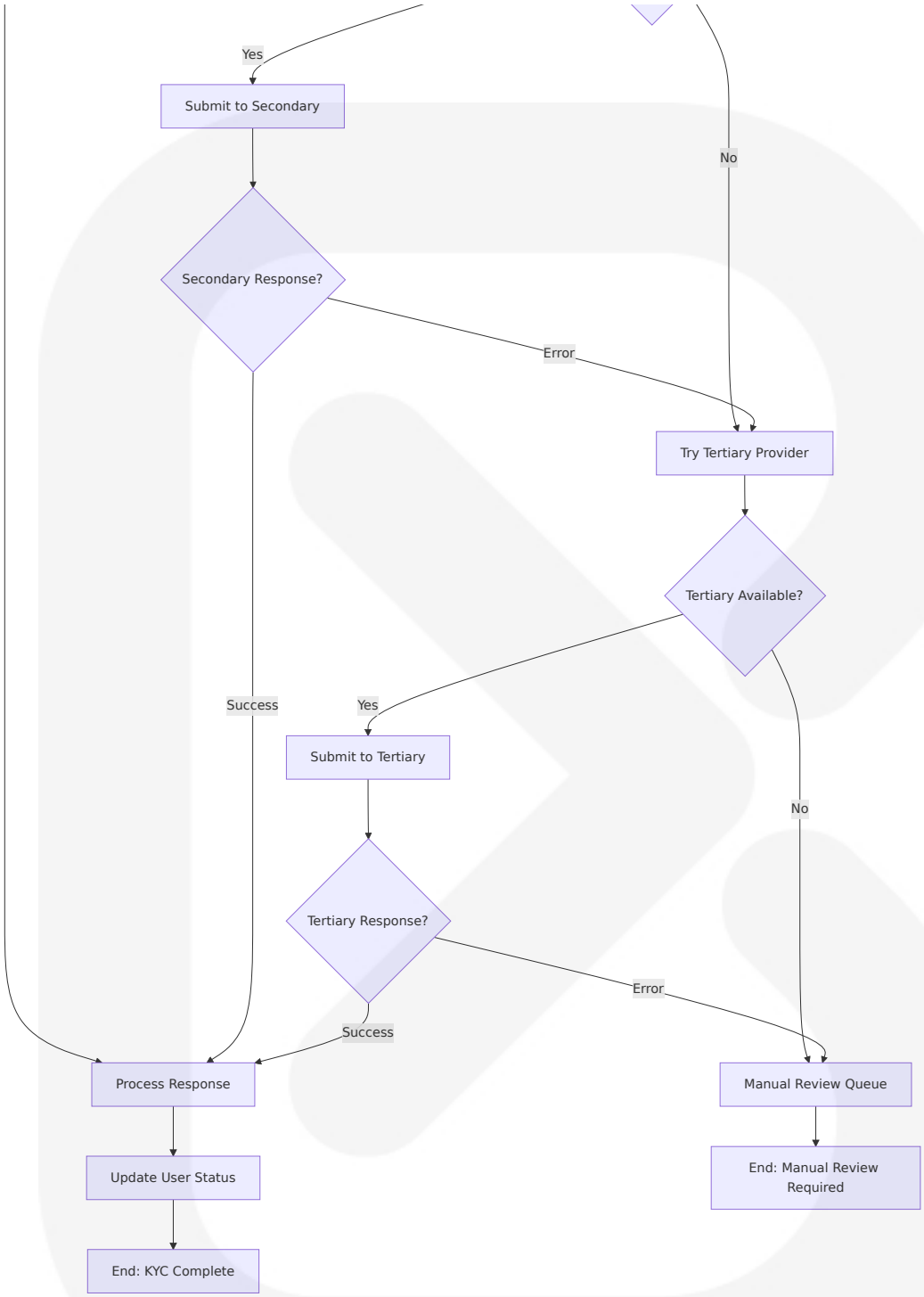
Recovery Strategies:

- Deadlock resolution: Random exponential backoff (100ms-5s)
- Connection pooling: Automatic connection recycling
- Transaction timeout: 30 seconds for complex operations
- Critical error escalation: Immediate alert to operations team

4.2.2 Compliance-Specific Error Handling

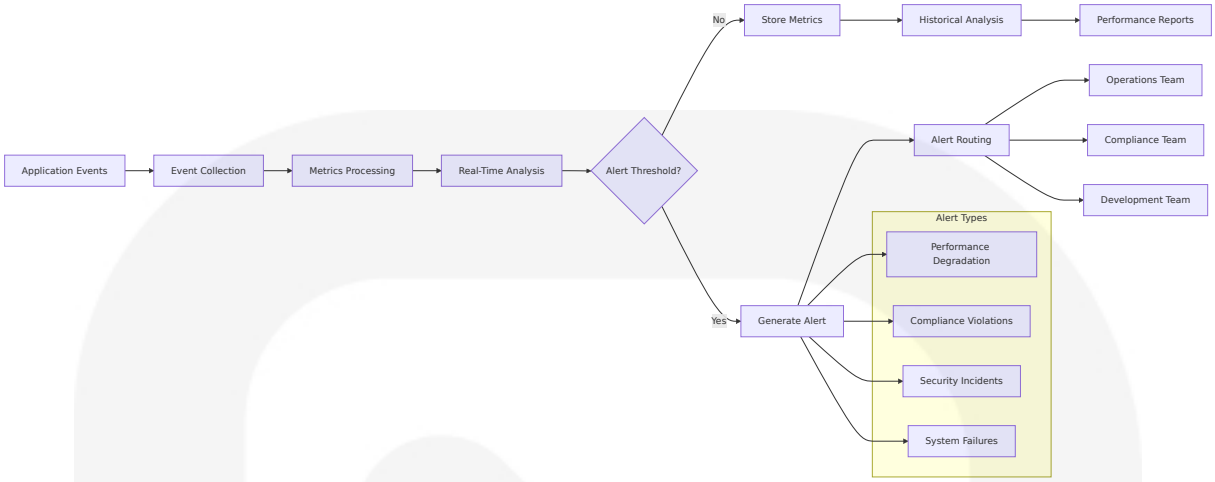
4.2.2.1 KYC Provider Failure Recovery


KYC Request Initiated
Submit to Primary Provider
Provider Response?
Success
Timeout
Error
Categorize Error
Rate Limit Error?
Yes
No
Implement Backoff
Authentication Error?
Yes
No
Queue for Later Processing
Refresh API Credentials
End: Deferred Processing
Refresh Success?
Yes
No
Log Timeout Event
Service Unavailable?
No
Yes
Escalate Unknown Error
Try Secondary Provider
End: System Error
Secondary Available?



4.3 PERFORMANCE AND MONITORING FLOWS

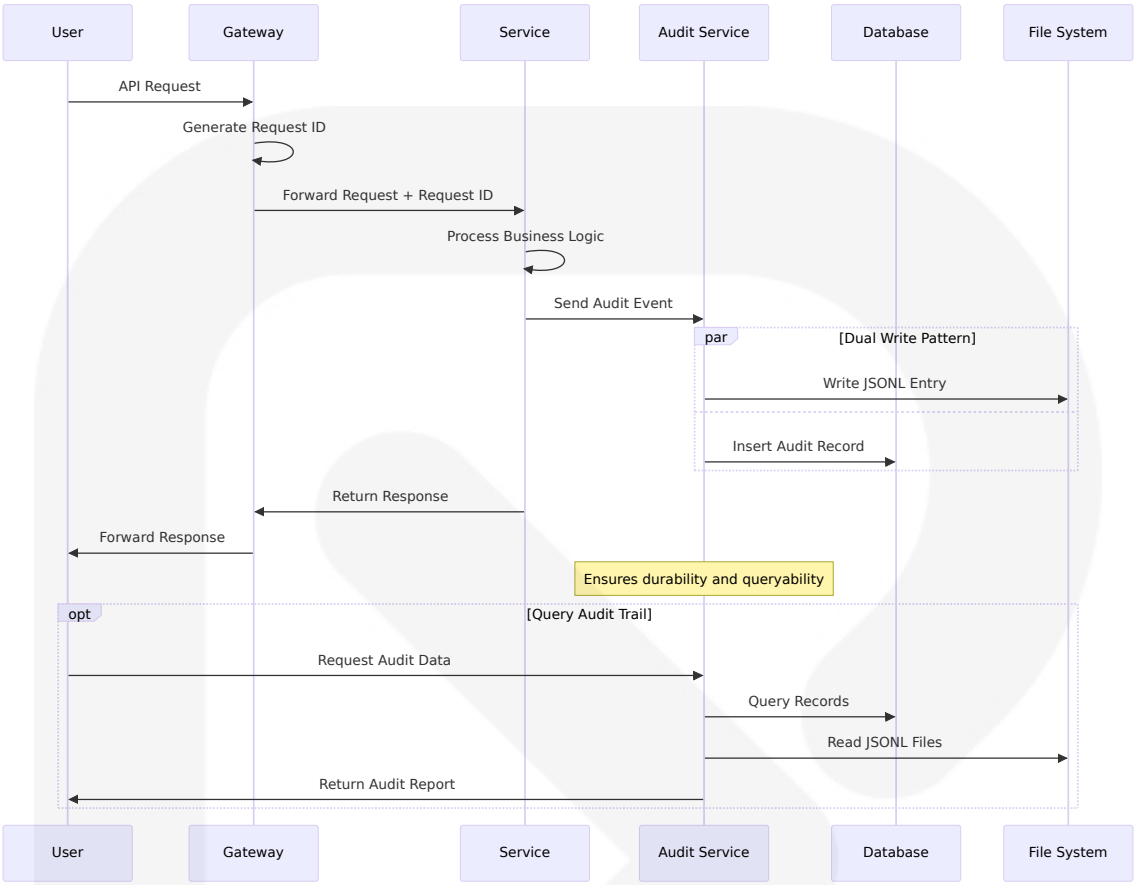
4.3.1 Real-Time Monitoring Pipeline



Key Performance Indicators:

- API response time p99 < 200ms
- Transaction throughput > 1000 TPS
- KYC processing time < 30 seconds
- System uptime 99.99% SLA
- Compliance rule evaluation < 100ms

4.3.2 Audit Trail Processing Flow

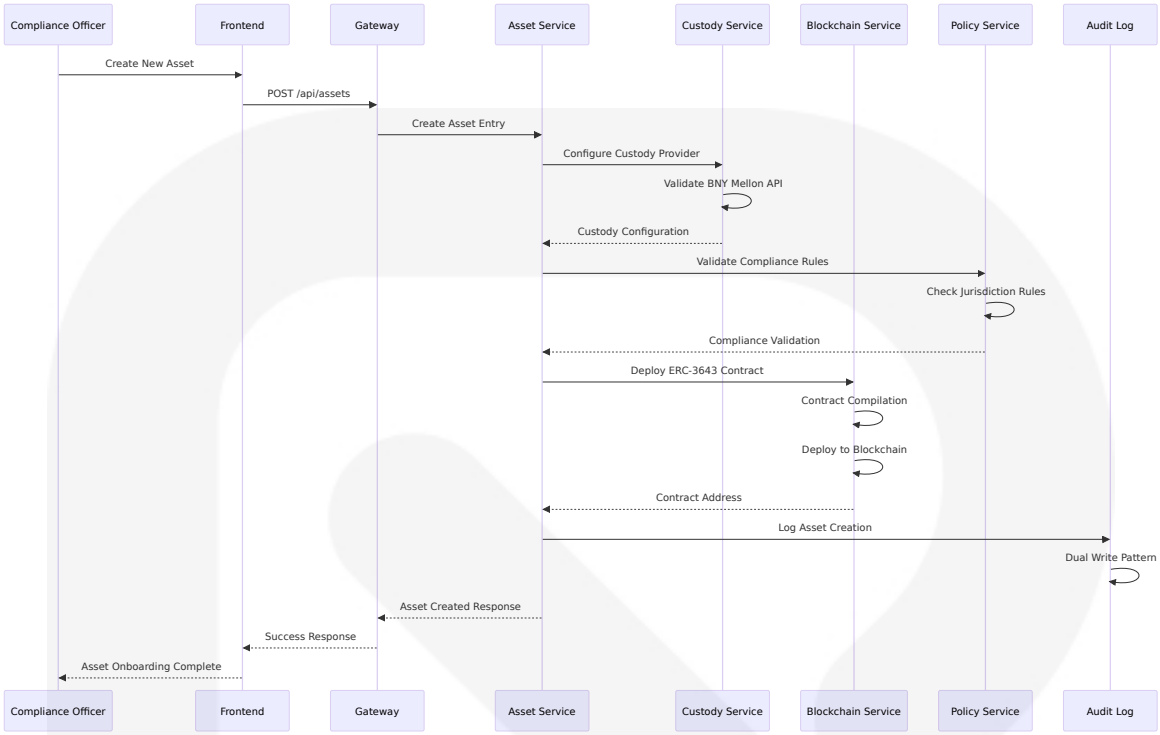


Audit Event Structure:

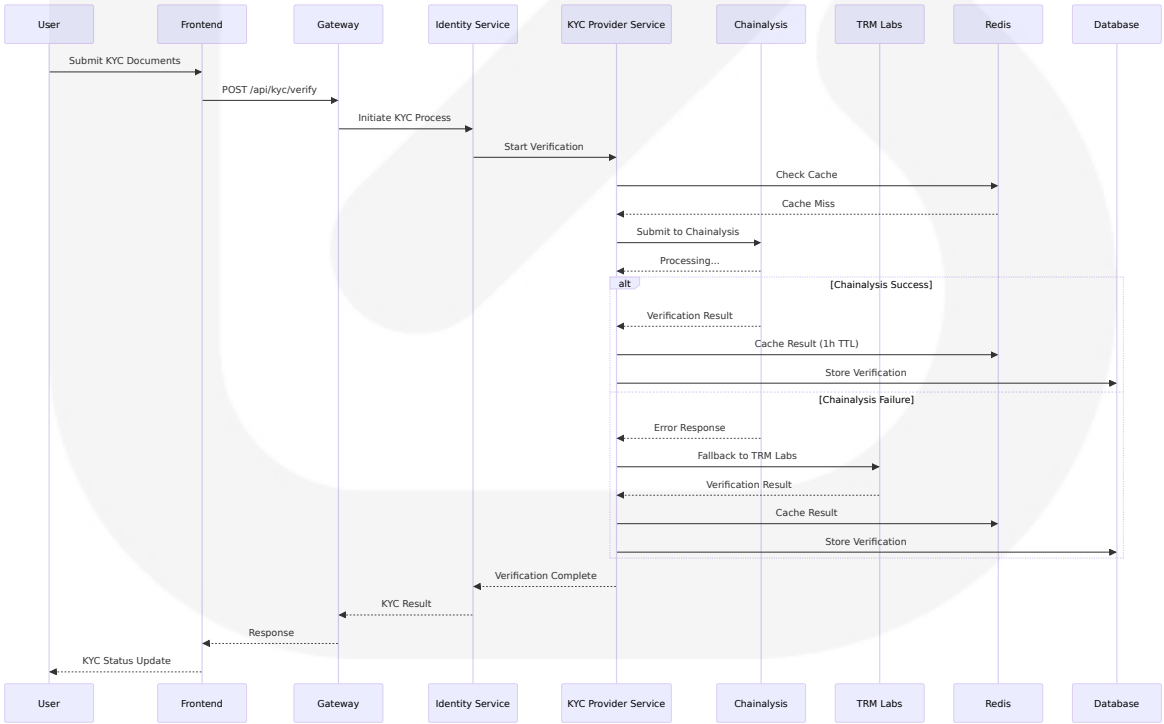
- Request ID for tracing across services
- User ID and organization context
- Service and endpoint identification
- Request/response payload hashing
- Timestamp with nanosecond precision
- Compliance context and decision factors

4.4 INTEGRATION SEQUENCE DIAGRAMS

4.4.1 Asset Onboarding Integration Sequence

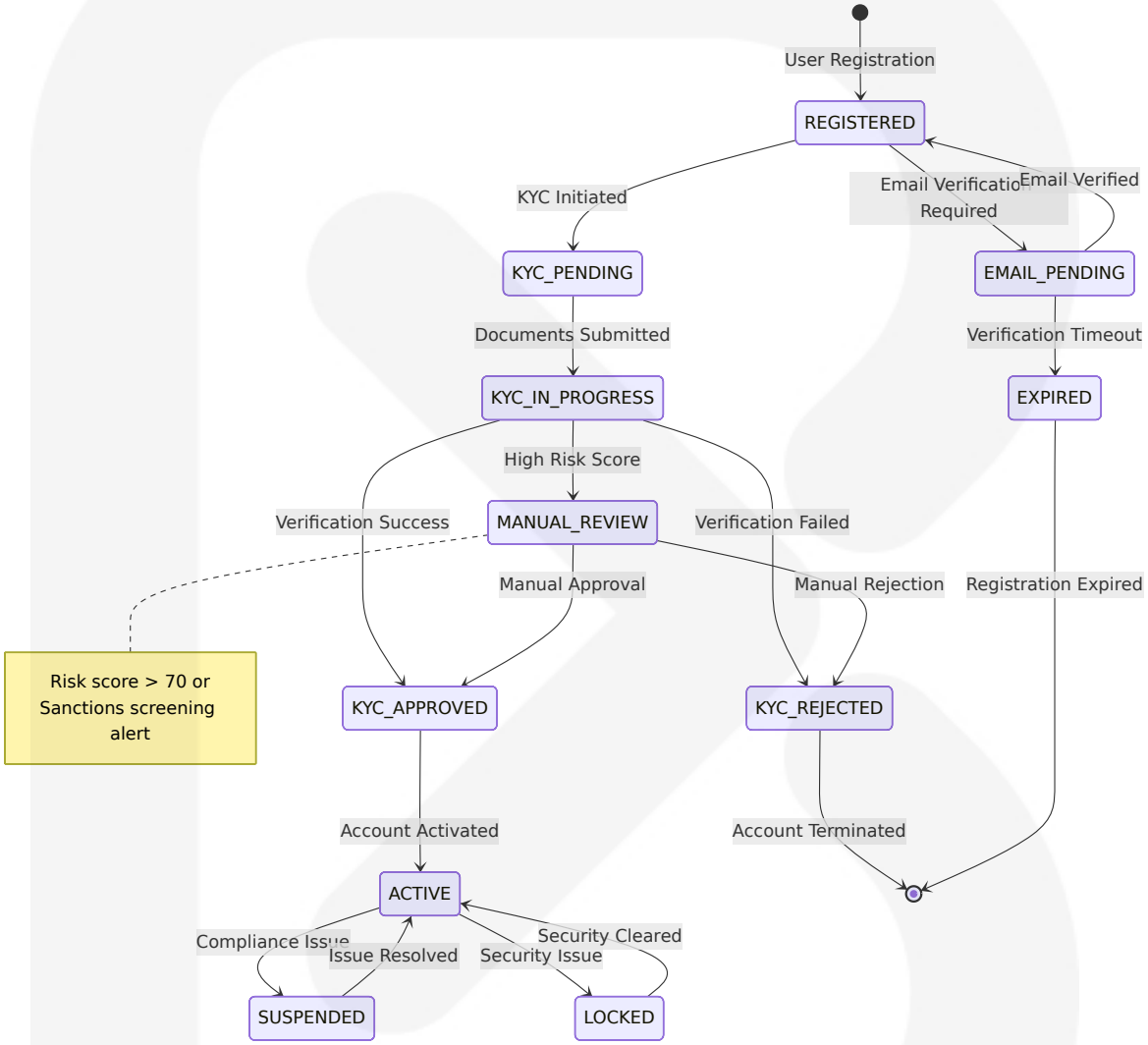


4.4.2 Multi-Provider KYC Integration Sequence

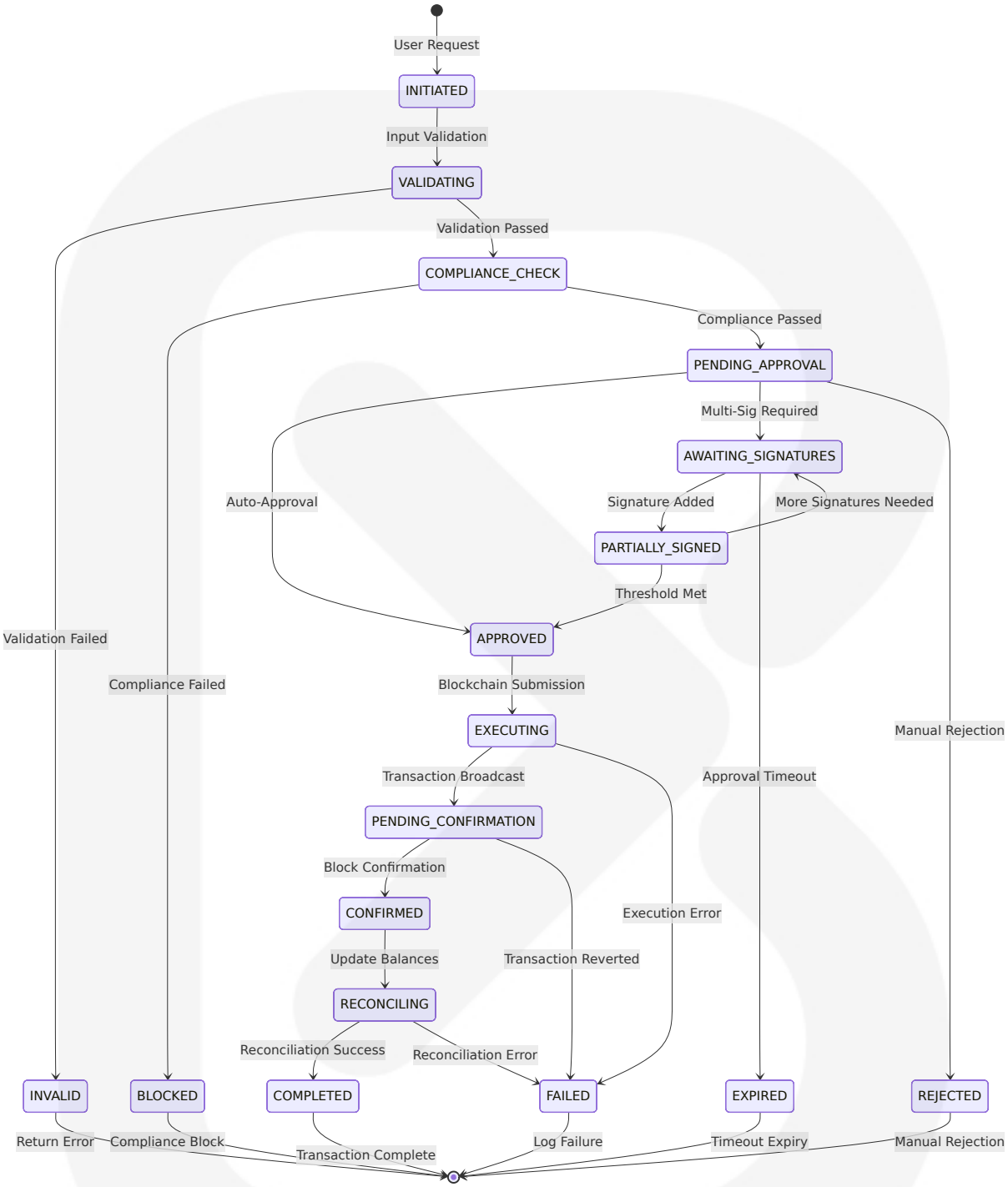


4.5 STATE TRANSITION DIAGRAMS

4.5.1 User Account State Transitions



4.5.2 Transaction Processing States



References

System Architecture Components Referenced:

- `services/gateway/` - API gateway routing and rate limiting implementation
- `services/identity-service/` - JWT authentication and session management
- `services/compliance-service/` - Rules engine and screening logic
- `services/kyc-provider/` - Multi-provider orchestration and caching
- `services/blockchain-service/` - Smart contract deployment and transaction management
- `services/audit-log-writer/` - Dual-write audit pattern implementation
- `services/regulatory-reporting/` - Report generation and templating
- `services/policy-service/` - YAML-driven policy management
- `packages/database/` - PostgreSQL schemas and transaction patterns
- `apps/compliance-dashboard/` - React frontend with role-based access

External Integration Points:

- Chainalysis, TRM Labs, Jumio, Onfido APIs for KYC verification
- BNY Mellon custody provider integration
- Ethereum, Polygon, Solana blockchain networks
- Redis caching layer with TTL-based expiration
- Stripe payment processing integration

Technical Specifications Retrieved:

- System Overview (Section 1.2) - High-level architecture and workflows
- Feature Catalog (Section 2.1) - Complete feature specifications and dependencies
- Functional Requirements (Section 2.2) - Detailed acceptance criteria and validation rules

5. SYSTEM ARCHITECTURE

5.1 HIGH-LEVEL ARCHITECTURE

5.1.1 System Overview

5.1.1.1 Architecture Style and Rationale

Veria implements a **microservices architecture with service mesh pattern**, designed specifically for the stringent requirements of financial compliance and regulatory technology. The architecture prioritizes data integrity, audit capability, and regulatory compliance while maintaining high performance and scalability.

Service Mesh Pattern: All backend services (ports 4001-4005) are accessed through a central Gateway service on port 4000, providing centralized routing, authentication, rate limiting, and monitoring. This pattern ensures consistent security policies, simplified client communication, and comprehensive request tracking across all services.

Event-Driven Components: The system incorporates real-time compliance monitoring and screening capabilities through Redis pub/sub messaging and asynchronous processing pipelines. This enables immediate compliance decision-making and automated risk assessment workflows.

Polyglot Implementation: While TypeScript/Node.js serves as the primary technology stack, Python middleware components handle specialized blockchain operations and compliance logic, leveraging the best tools for specific problem domains.

5.1.1.2 Architectural Principles

Financial Data Integrity: Every component implements ACID-compliant operations with comprehensive audit trails, ensuring regulatory compliance and data consistency across all financial transactions and compliance decisions.

Zero-Trust Security: JWT-based authentication with role-based access control (RBAC) ensures that every request is authenticated and authorized, with no implicit trust between services.

Performance-First Design: Fastify framework selection delivers up to 30,000 requests per second with built-in schema validation, supporting the platform's <200ms API response time requirements.

Resilience and Failover: Multi-provider integration patterns with automatic fallback mechanisms ensure system availability even when external compliance services experience outages.

5.1.1.3 System Boundaries and Interfaces

Internal Service Boundaries: Eleven distinct microservices handle specialized domains (identity, compliance, audit, KYC, blockchain, etc.) with clear separation of concerns and well-defined API contracts.

External Integration Points: The system interfaces with multiple external providers including KYC services (Chainalysis, TRM Labs, Jumio, Onfido), custody providers (BNY Mellon), payment processors (Stripe), and blockchain networks (Ethereum, Polygon, Solana).

User Interface Boundaries: Two distinct frontend applications serve different user roles - a compliance dashboard (React/Vite) for operational users and a main frontend (Next.js) for external stakeholders.

5.1.2 Core Components Table

Component Name	Primary Responsibility	Key Dependencies	Integration Points	Critical Considerations
Gateway Service	API routing, authentication, rate limiting	Redis, downstream services	All client requests, service mesh hub	Single point of failure, requires high availability

Component Name	Primary Responsibility	Key Dependencies	Integration Points	Critical Considerations
Identity Service	User authentication, RBAC, session management	PostgreSQL, Redis, JWT tokens	All authenticated endpoints	Token security, session management, WebAuthn support
Compliance Service	Rule evaluation, screening, monitoring	PostgreSQL, Redis, Qdrant	KYC Provider, Audit services	Real-time decision making, regulatory updates
KYC Provider Service	Multi-provider orchestration, verification	External KYC APIs, Redis cache	Compliance, Identity services	Provider availability, fallback mechanisms

5.1.3 Data Flow Description

5.1.3.1 Primary Data Flows

Authentication Flow: User credentials flow through the Gateway to the Identity Service, which validates against PostgreSQL user records and generates JWT tokens. Session state is maintained in Redis with 7-day TTL, enabling distributed session management across service instances.

Compliance Decision Flow: Asset or investor data flows from client applications through the Gateway to the Compliance Service, which evaluates YAML-driven compliance rules. Results are cached in Redis with 60-second TTL to optimize repeated evaluations. Decisions trigger audit log entries through the dual-write pattern.

KYC Processing Flow: Document uploads are routed to the KYC Provider Service, which orchestrates verification across multiple external providers (Chainalysis, TRM Labs, Jumio, Onfido) with intelligent fallback. Results are

cached in Redis for 1-hour TTL and stored permanently in PostgreSQL with complete audit trails.

5.1.3.2 Integration Patterns

Request-Response Pattern: Synchronous API calls for real-time compliance decisions and user authentication, ensuring immediate feedback for critical operations.

Event-Driven Pattern: Asynchronous processing for heavy operations like compliance report generation and blockchain transaction monitoring, preventing blocking of user-facing operations.

Cache-Aside Pattern: Redis caching for frequently accessed data (compliance decisions, KYC results, user sessions) with configurable TTL values optimized for each data type's staleness tolerance.

5.1.3.3 Data Transformation Points

API Gateway Layer: Request/response transformation including rate limiting headers, request ID injection, and error standardization across all services.

Compliance Engine: Business rule evaluation transforms raw compliance data into actionable decisions with risk scores and recommended actions.

Report Generation: Multi-format transformation (PDF, CSV, JSON) of compliance data for regulatory reporting requirements.

5.1.4 External Integration Points

System Name	Integration Type	Data Exchange Pattern	Protocol/ Format	SLA Requirements
Chainalysis	KYC/AML Screening	Request/Response	REST API/JSON	95% availability, <5s response
BNY Mellon	Custody Services	Batch/Real-time	REST API/JSON	99.9% availability, <2s response
Ethereum Network	Blockchain Operations	Event-driven	JSON-RPC/ WebSocket	Network-dependent availability
Qdrant	Vector Search	Request/Response	HTTP/gRPC	99.5% availability, <100ms response

5.2 COMPONENT DETAILS

5.2.1 Gateway Service Architecture

5.2.1.1 Purpose and Responsibilities

The Gateway Service serves as the single entry point for all client requests, implementing the API Gateway pattern with comprehensive traffic management, security enforcement, and observability features. It routes requests to appropriate backend services while enforcing rate limits, authentication, and request/response transformations.

5.2.1.2 Technologies and Frameworks

Fastify v4.25.2: High-performance web framework delivering up to 30,000 requests per second with built-in schema validation and plugin ecosystem support.

Redis Integration: Rate limiting implementation with sliding window algorithm (100 requests per 60-second window per IP address) and distributed session validation.

Request ID Propagation: Automatic x-request-id header injection enabling distributed tracing across the entire service mesh.

5.2.1.3 Key Interfaces and APIs

Health Check Aggregation: `/health` endpoint aggregates health status from all downstream services, providing comprehensive system health visibility for load balancers and monitoring systems.

Service Routing: Dynamic request routing based on URL patterns to appropriate backend services (identity-service:4001, compliance-service:4002, etc.).

Error Handling: Centralized error response formatting with consistent structure across all API endpoints.

5.2.1.4 Scaling Considerations

Stateless Design: No local state storage enables horizontal scaling across multiple instances with load balancer distribution.

Connection Pooling: HTTP keep-alive connections to backend services reduce connection overhead and improve performance.

Circuit Breaker Pattern: Automatic service health monitoring with request blocking to unhealthy services, preventing cascade failures.

5.2.2 Identity Service Architecture

5.2.2.1 Purpose and Responsibilities

The Identity Service manages user authentication, authorization, and session state with support for JWT tokens, WebAuthn/passkeys, and comprehensive RBAC with 7 hierarchical roles (SUPER_ADMIN, ADMIN, COMPLIANCE_OFFICER, INVESTOR, INSTITUTION, ISSUER, VIEWER).

5.2.2.2 Technologies and Frameworks

PostgreSQL Integration: User account storage with UUID primary keys, JSONB columns for flexible profile data, and optimized indexes for authentication queries.

JWT Implementation: HS256 signing with dual-token pattern (15-minute access tokens, 7-day refresh tokens) ensuring security while minimizing authentication overhead.

WebAuthn Support: Passwordless authentication implementation supporting FIDO2 security keys and biometric authentication for enhanced security.

5.2.2.3 Data Persistence Requirements

Session Management: Redis storage with 7-day sliding expiration for active sessions, enabling distributed session validation across service instances.

Token Blacklisting: Redis-based revoked token tracking with 24-hour TTL, preventing usage of compromised or revoked tokens.

User Profile Storage: PostgreSQL persistence for user account data, organization associations, and permission mappings.

5.2.3 Compliance Service Architecture

5.2.3.1 Purpose and Responsibilities

The Compliance Service implements the core regulatory logic including screening engines, compliance rule evaluation, multi-format reporting generation, and real-time monitoring with alerting capabilities.

5.2.3.2 Technologies and Frameworks

YAML-Driven Rules Engine: Declarative compliance policy definition enabling business users to modify compliance logic without code changes.

Qdrant Integration: Vector similarity search for semantic compliance document analysis and regulatory text matching.

Multi-Format Reporting: Template-based generation supporting PDF, CSV, and JSON formats using Handlebars templating engine.

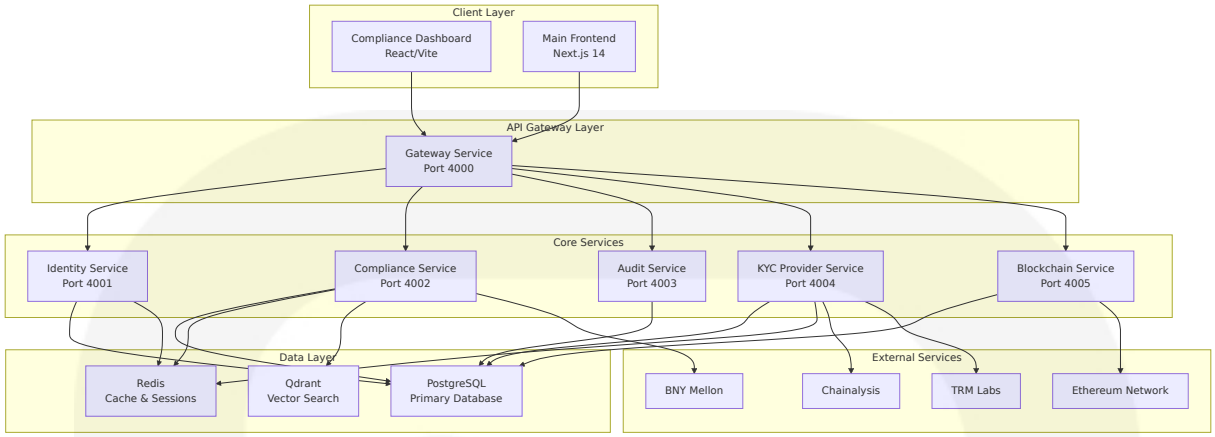
5.2.3.3 Key Interfaces and APIs

Rule Evaluation API: Real-time compliance decision-making with configurable caching (300-second TTL for policy results).

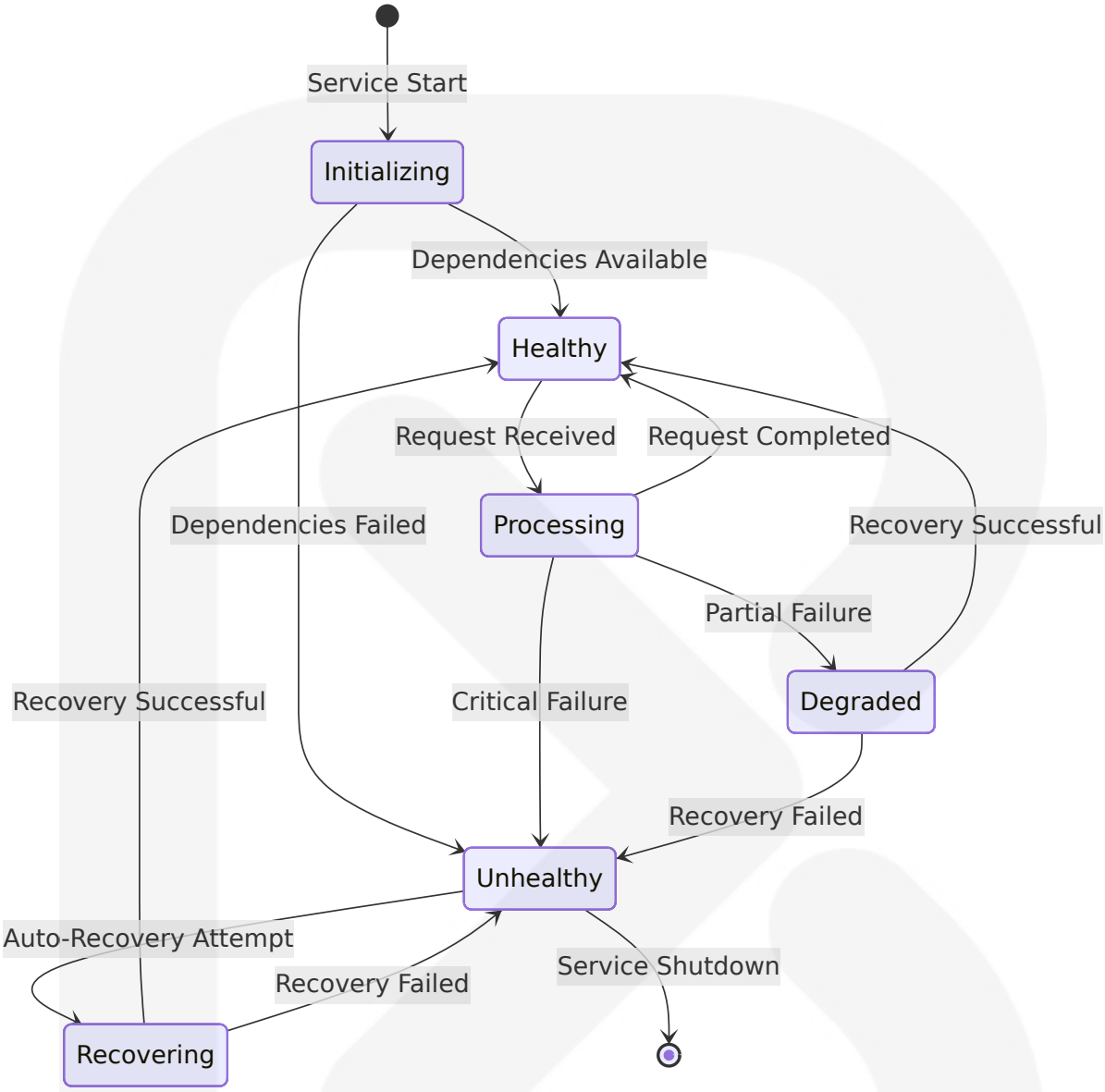
Screening Integration: Automated sanctions screening and risk assessment with audit trail generation.

Report Generation API: Asynchronous compliance report creation with progress tracking and secure download URL generation.

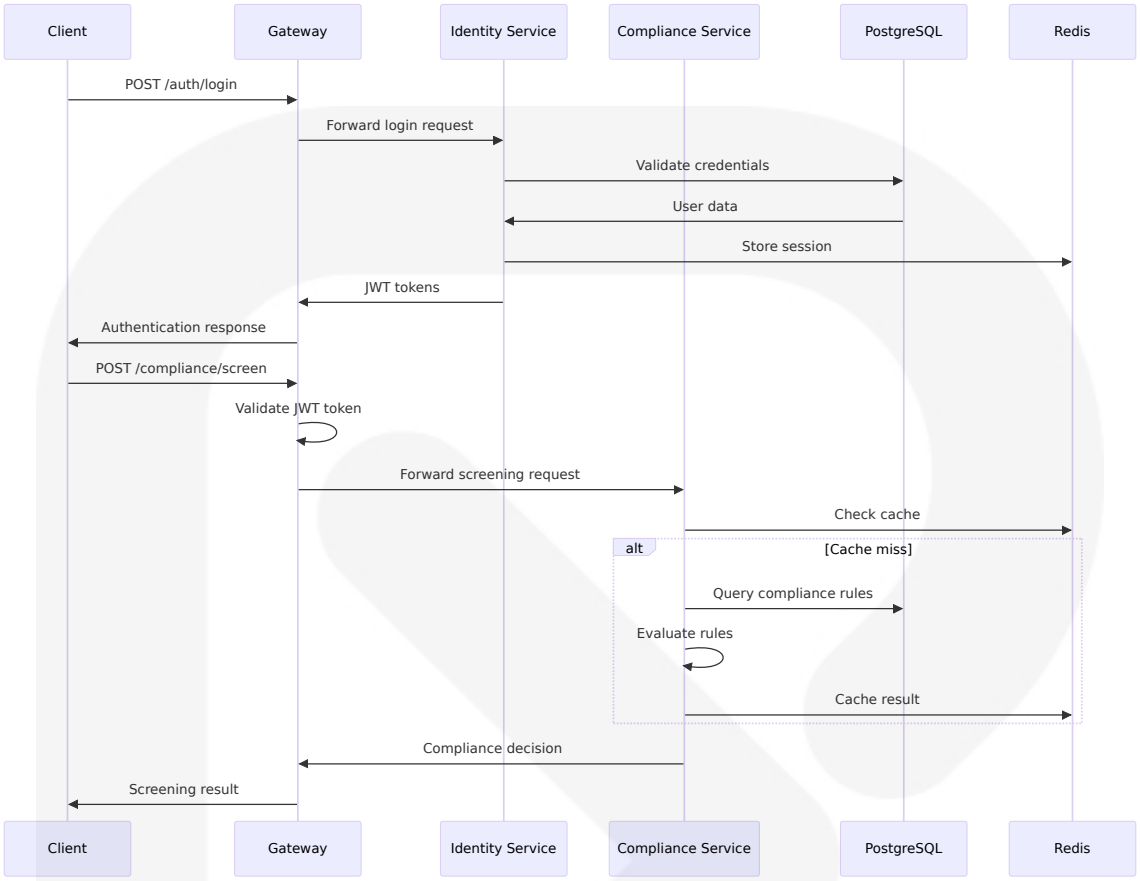
5.2.4 Component Interaction Diagram



5.2.5 Service State Transition Diagram



5.2.6 Key Flow Sequence Diagram



5.3 TECHNICAL DECISIONS

5.3.1 Architecture Style Decisions

5.3.1.1 Microservices vs Monolith Decision

Decision: Microservices architecture with service mesh pattern

Rationale: Financial compliance requires domain-specific expertise and independent scaling. Microservices enable:

- **Regulatory Isolation**: Separate compliance domains (KYC, AML, audit) can evolve independently as regulations change
- **Technology Optimization**: Python for blockchain operations, Node.js for API services, optimizing each domain

- **Deployment Independence**: Critical services can be updated without affecting the entire system
- **Team Autonomy**: Different teams can own compliance, identity, and audit domains independently

Tradeoffs: Increased operational complexity and distributed system challenges versus regulatory flexibility and scalability benefits.

5.3.1.2 Service Mesh Pattern Selection

Decision: Centralized Gateway service routing pattern

Rationale:

- **Security Centralization**: Single point for authentication, rate limiting, and audit logging
- **Compliance Requirements**: Centralized request/response logging for regulatory audit trails
- **Service Discovery**: Simplified client integration with single endpoint
- **Cross-Cutting Concerns**: Consistent error handling, monitoring, and observability

Implementation: All client requests route through Gateway (port 4000) to backend services (ports 4001-4005).

5.3.2 Communication Pattern Choices

5.3.2.1 Synchronous vs Asynchronous Communication

Operation Type	Pattern	Justification	Implementation
User Authentication	Synchronous	Immediate feedback required	JWT validation in Gateway

Operation Type	Pattern	Justification	Implementation
Compliance Decisions	Synchronous	Real-time risk assessment	Direct API calls with caching
Report Generation	Asynchronous	Long-running processes	Job queues with progress tracking
Audit Logging	Dual-write	Immediate + eventual consistency	Filesystem + PostgreSQL

5.3.2.2 API Design Decisions

REST API Selection: Chosen over GraphQL for financial services due to:

- **Regulatory Compliance**: Standard HTTP methods align with audit requirements
- **Caching Simplicity**: HTTP-based caching works with existing infrastructure
- **Tool Ecosystem**: Better support for security scanning and compliance tools
- **Team Familiarity**: Reduced learning curve for financial technology teams

5.3.3 Data Storage Solution Rationale

5.3.3.1 PostgreSQL as Primary Database

Decision: PostgreSQL v14 for all transactional data storage

Rationale:

- **ACID Compliance**: Financial transactions require strict consistency guarantees
- **Advanced Indexing**: GIN indexes for JSONB columns support flexible compliance data structures

- **Audit Requirements**: Write-ahead logging provides complete transaction history
- **Ecosystem Maturity**: Excellent tooling for backup, monitoring, and performance optimization

Configuration: Connection pooling with QueuePool (pool_size=20, max_overflow=10) optimized for concurrent compliance operations.

5.3.3.2 Multi-Tier Caching Strategy

Cache Type	Technology	TTL	Use Case	Rationale
Session Cache	Redis	7 days	User authentication state	Distributed session management
Compliance Decisions	Redis	60 seconds	Rule evaluation results	Reduce compliance engine load
KYC Results	Redis	1 hour	Provider verification status	Minimize external API calls
Policy Rules	Redis	300 seconds	Compliance rule definitions	Optimize rule evaluation performance

5.3.4 Security Mechanism Selection

5.3.4.1 JWT Authentication Strategy

Decision: Dual-token JWT implementation (15-minute access, 7-day refresh)

Security Benefits:

- **Short Access Window**: Minimizes exposure if tokens are compromised

- **Refresh Rotation**: Automatic token renewal prevents long-term compromise
- **Stateless Validation**: Distributed service authentication without centralized session store
- **WebAuthn Integration**: Passwordless authentication for enhanced security

Implementation: HS256 signing with Redis-based token blacklisting for immediate revocation.

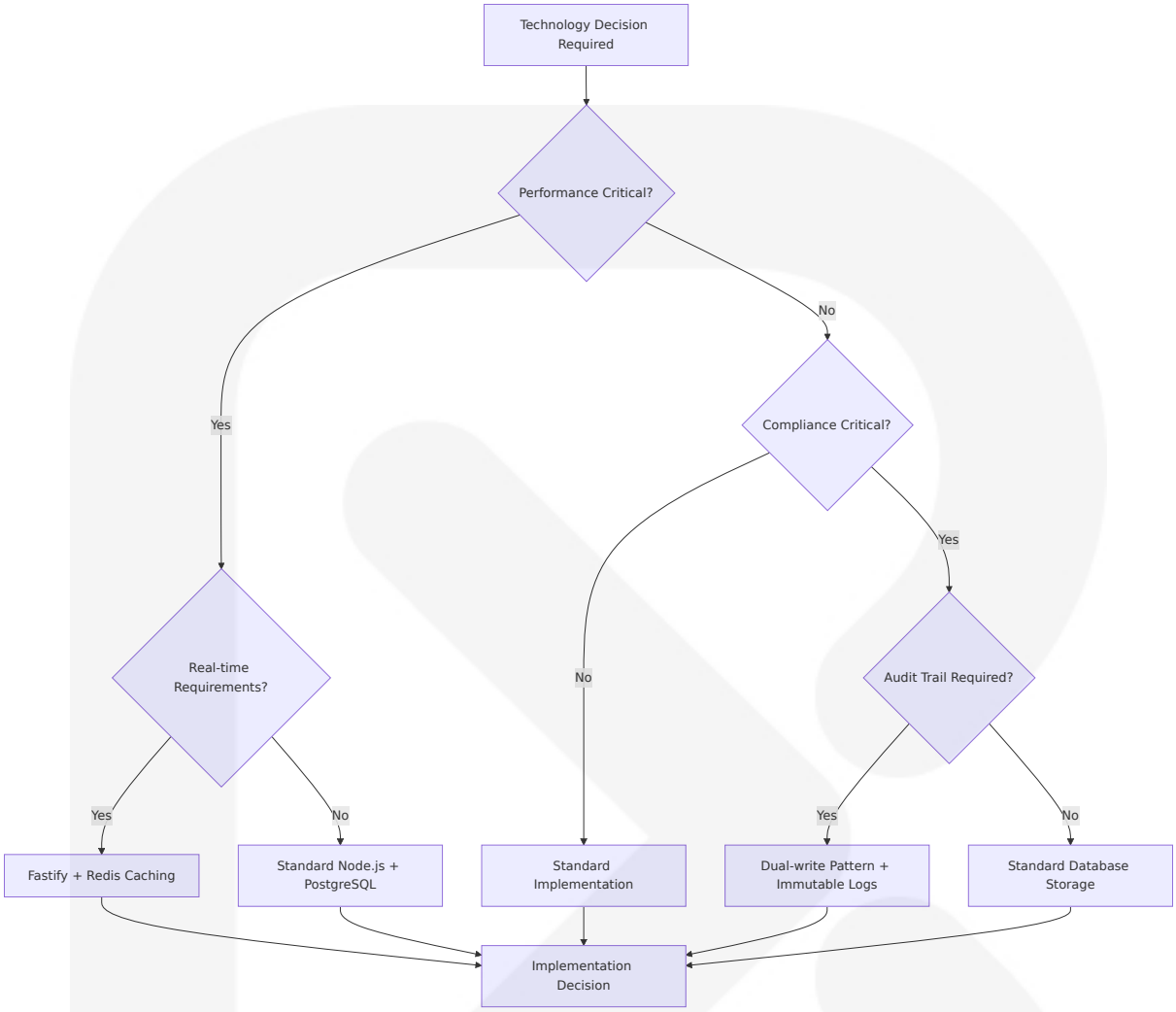
5.3.4.2 Role-Based Access Control (RBAC)

Decision: Seven-tier hierarchical role system

Role Hierarchy:

1. **SUPER_ADMIN**: System administration and configuration
2. **ADMIN**: Organization management and user administration
3. **COMPLIANCE_OFFICER**: Compliance rule management and oversight
4. **INVESTOR**: Investment management and portfolio access
5. **INSTITUTION**: Institutional client operations
6. **ISSUER**: Asset issuance and management
7. **VIEWER**: Read-only access to authorized data

5.3.5 Technology Decision Tree



5.4 CROSS-CUTTING CONCERNS

5.4.1 Monitoring and Observability Approach

5.4.1.1 Application Performance Monitoring

Distributed Tracing: Request ID propagation (x-request-id header) enables end-to-end request tracking across all services. Each service adds its processing information to the trace, providing complete visibility into request flows and performance bottlenecks.

Metrics Collection: Prometheus-compatible metrics exposed by each service including:

- Request/response times with percentile distributions
- Error rates and status code distributions
- Database connection pool utilization
- Redis cache hit/miss ratios
- External provider response times and availability

Service Health Monitoring: Comprehensive health checks aggregated through the Gateway service, providing both individual service status and overall system health visibility.

5.4.1.2 Business Metrics Monitoring

Compliance Metrics: Real-time tracking of compliance decision rates, screening throughput, and regulatory rule evaluation performance to ensure SLA compliance.

Financial Transaction Monitoring: Transaction processing rates, settlement times, and failure rates with automated alerting for anomalies that could impact financial operations.

User Experience Metrics: Authentication success rates, session duration tracking, and user workflow completion rates to optimize compliance processes.

5.4.2 Logging and Tracing Strategy

5.4.2.1 Structured Logging Implementation

Pino Logger Integration: High-performance JSON logging with configurable log levels across all services. Production deployments use structured JSON format for automated log analysis and compliance reporting.

Log Correlation: Request IDs enable log aggregation across services for complete request lifecycle analysis. Each log entry includes service name, timestamp, request ID, and contextual business data.

Compliance Log Requirements: Specialized audit logging for regulatory compliance with immutable log storage and hash verification for tamper detection.

5.4.2.2 Dual-Write Audit Pattern

Filesystem Logging: Synchronous JSONL file append to `.audit-data/audit.log` for immediate audit trail availability with minimal latency impact.

Database Persistence: Asynchronous PostgreSQL storage for queryable audit data with full-text search capabilities and long-term retention policies.

Hash Verification: SHA-256 hash calculation for audit log integrity verification, enabling detection of unauthorized modifications to audit trails.

5.4.3 Error Handling Patterns

5.4.3.1 Service-Level Error Handling

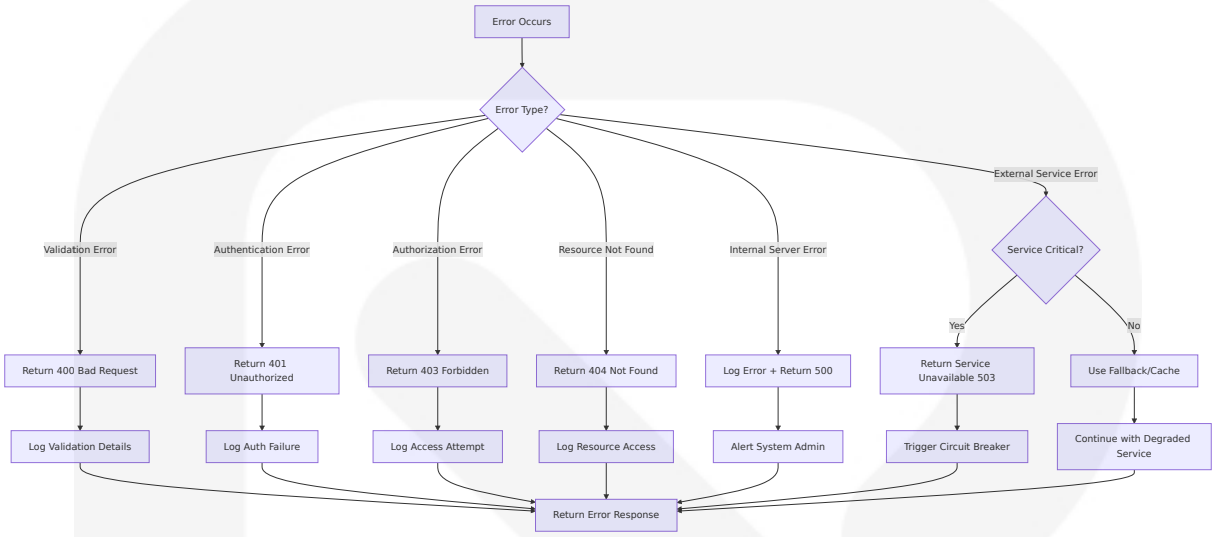
Circuit Breaker Pattern: Automatic service degradation when downstream dependencies become unavailable, preventing cascade failures across the service mesh.

Retry Mechanisms: Configurable retry policies with exponential backoff for external service integrations, particularly important for KYC provider communications.

Graceful Degradation: Fallback mechanisms that maintain core functionality when non-critical services are unavailable, ensuring

compliance operations continue during partial outages.

5.4.3.2 Error Handling Flow Diagram



5.4.4 Authentication and Authorization Framework

5.4.4.1 Multi-Factor Authentication

WebAuthn Integration: FIDO2-compliant passwordless authentication supporting hardware security keys and biometric authentication for enhanced security in financial environments.

JWT Token Management: Dual-token strategy with short-lived access tokens (15 minutes) and longer-lived refresh tokens (7 days) balancing security and user experience.

Session Management: Redis-based distributed session storage with sliding expiration (7-day TTL) enabling session persistence across service instances.

5.4.4.2 Role-Based Authorization

Permission Matrix: Fine-grained permission system mapping user roles to specific API endpoints and data access levels, ensuring compliance with financial privacy regulations.

Dynamic Role Assignment: Support for multiple organizational contexts where users may have different roles in different organizations or asset classes.

Audit Trail Integration: All authentication and authorization events logged for compliance auditing and security incident investigation.

5.4.5 Performance Requirements and SLAs

5.4.5.1 Service Level Objectives

Component	Availability SLA	Response Time SLA	Throughput SLA	Error Rate SLA
Gateway Service	99.99%	<50ms (p95)	10,000+ req/sec	<0.1%
Identity Service	99.95%	<100ms (p95)	5,000+ auth/sec	<0.5%
Compliance Service	99.90%	<200ms (p95)	1,000+ decisions/sec	<1.0%
KYC Provider Service	99.50%	<5s (p95)	100+ verifications/sec	<2.0%

5.4.5.2 Performance Optimization Strategies

Connection Pooling: PostgreSQL connection pools optimized for concurrent operations (pool_size=20, max_overflow=10) with monitoring for pool exhaustion.

Caching Strategy: Multi-tier Redis caching with domain-specific TTL values optimized for data staleness tolerance and performance requirements.

Query Optimization: Database indexes optimized for common compliance queries with regular performance analysis and optimization cycles.

5.4.6 Disaster Recovery Procedures

5.4.6.1 Backup and Recovery Strategy

Database Backup: Daily PostgreSQL backups with point-in-time recovery capabilities and 30-day retention policy for regulatory compliance.

Redis Persistence: Redis AOF (Append Only File) persistence enabled for cache and session recovery with automatic failover to secondary instances.

Audit Log Preservation: Immutable audit log storage with offsite backup and long-term retention (7 years) meeting financial regulatory requirements.

5.4.6.2 Service Recovery Procedures

Automated Health Checks: Continuous service monitoring with automatic restart policies for failed instances and immediate alerting for manual intervention.

Circuit Breaker Recovery: Automatic circuit breaker reset with gradual traffic restoration to recovered services, preventing service overload during recovery.

Data Consistency Verification: Post-recovery validation procedures ensuring data integrity across all services and external integrations.

References

Technical Specification Sections Referenced

- `1.2 SYSTEM OVERVIEW` - Overall system context and business requirements
- `3.2 FRAMEWORKS & LIBRARIES` - Technology stack and framework decisions
- `3.5 DATABASES & STORAGE` - Data architecture and persistence strategies
- `3.6 DEVELOPMENT & DEPLOYMENT` - Infrastructure and deployment patterns
- `4.1 SYSTEM WORKFLOWS` - Business process flows informing architecture
- `2.5 NON-FUNCTIONAL REQUIREMENTS` - Performance and quality attributes

Repository Files Examined

- `package.json` - Root workspace configuration
- `docker-compose.yml` - Service orchestration configuration
- `Dockerfile` - Container build configuration
- `cloudrun.yaml` - Google Cloud Run deployment manifest
- `services/gateway/package.json` - Gateway service dependencies
- `services/compliance-service/package.json` - Compliance service configuration
- `services/identity-service/package.json` - Identity service dependencies
- `packages/auth-middleware/package.json` - Authentication middleware
- `packages/database/package.json` - Database configuration
- `.env.example` - Environment configuration template
- `.github/workflows/ci.yml` - CI pipeline configuration
- `.github/workflows/deploy-backend.yml` - Deployment automation

Repository Folders Analyzed

- `/` - Root repository structure and monorepo configuration
- `services/` - 11 microservices architecture overview
- `packages/` - Shared packages for cross-service functionality
- `apps/` - Frontend applications and user interfaces
- `infra/` - Infrastructure-as-code and deployment configurations
- `services/gateway/` - API gateway implementation details

- `services/compliance-service/` - Compliance engine architecture
- `services/identity-service/` - Authentication and authorization system
- `packages/auth-middleware/` - JWT and security middleware
- `packages/database/` - Database models and migration management

6. SYSTEM COMPONENTS DESIGN

6.1 CORE SERVICES ARCHITECTURE

6.1.1 SERVICE COMPONENTS

6.1.1.1 Service Boundaries and Responsibilities

Veria implements a **microservices architecture with service mesh pattern**, comprising 11 distinct services (9 active, 2 future placeholders) that follow domain-driven design principles. Each service maintains clear boundaries and specialized responsibilities within the compliance middleware ecosystem.

Active Service Components

Service Name	Port	Framework	Primary Responsibilities	Domain Expertise
Gateway Service	3001/4000	Fastify + Node.js ESM	API routing, authentication, rate limiting	Traffic orchestration
Identity Service	3002/4001	Fastify + TypeScript	JWT management, RBAC, WebAuthn, user/org CRUD	Identity & access management

Service Name	Port	Framework	Primary Responsibilities	Domain Expertise
Compliance Service	3004/4002	Fastify + TypeScript	Rule evaluation, screening, monitoring, reporting	Regulatory compliance
Policy Service	3003/4002	Fastify	Policy management, evaluation, simulation	Business rule engine

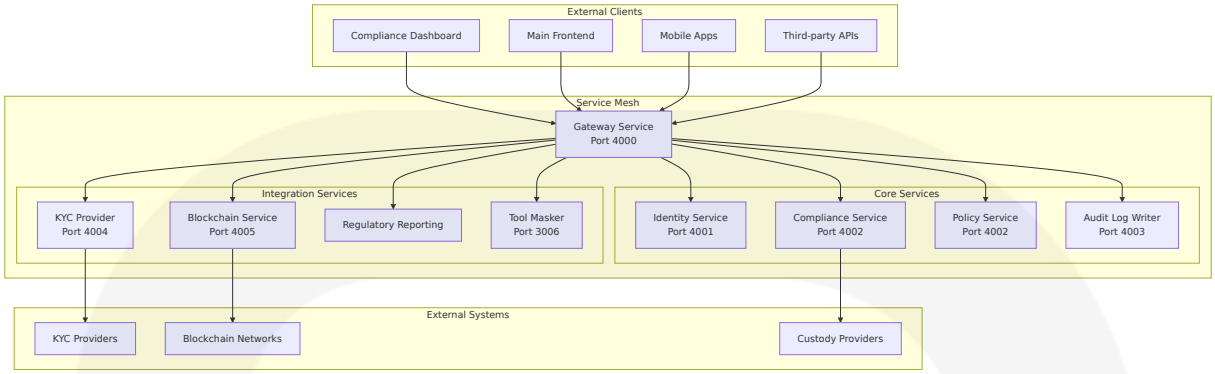
Specialized Service Components

Service Name	Port	Framework	Primary Responsibilities	Integration Focus
Audit Log Writer	3005/4003	Express	Dual-write audit trails (JSONL + PostgreSQL)	Compliance auditing
KYC Provider	3007/4004	Fastify + TypeScript	Multi-provider orchestration, verification caching	External KYC integration
Blockchain Service	N/A/4005	Fastify + Ethers v6	Token operations, identity registry, contract interaction	Blockchain integration
Regulatory Reporting	N/A	Fastify	SAR/CTR generation, scheduled jobs, multi-format export	Regulatory reporting

6.1.1.2 Inter-Service Communication Patterns

Service Mesh Gateway Pattern

All external requests flow through the Gateway service, implementing a centralized traffic management approach:



Communication Protocols and Standards

HTTP/REST Communication: All inter-service communication utilizes synchronous HTTP/REST calls with JSON payloads. The Gateway service implements request proxying with standardized headers:

- **Request ID Propagation**: `x-request-id` header ensures request traceability across service boundaries
- **Content-Type Standardization**: All services communicate via `application/json`
- **Authentication Headers**: JWT tokens propagated through `Authorization` header
- **Rate Limiting Headers**: `X-RateLimit-*` headers for client awareness

Request Transformation Pipeline: The Gateway service implements request/response transformation including header injection, rate limiting enforcement, and error standardization.

6.1.1.3 Service Discovery Mechanisms

Static Service Discovery

Veria implements **static service discovery** through environment-based configuration, optimized for the platform's stable microservices topology:

Service	Environment Variable	Development URL	Production URL
Identity Service	`IDENTITY_URL`	http://localhost:3002	Internal service mesh
Policy Service	`POLICY_URL`	http://localhost:3003	Internal service mesh
Compliance Service	`COMPLIANCE_URL`	http://localhost:3004	Internal service mesh
Audit Log Writer	`AUDIT_URL`	http://localhost:3005	Internal service mesh

Service Registry Pattern: The Gateway service maintains service endpoint mappings configured at startup, eliminating runtime service discovery overhead and ensuring deterministic routing behavior.

6.1.1.4 Load Balancing Strategy

Cloud-Native Load Balancing

Google Cloud Run Load Balancing: Production deployments leverage Cloud Run's built-in load balancing capabilities:

- **Container Concurrency**: 100 concurrent requests per container instance
- **Auto-scaling**: Based on CPU utilization and request volume
- **Health Check Integration**: `/health` endpoint monitoring with automatic instance replacement

Development Load Balancing: Docker Compose orchestration with single-instance services for simplified development workflows.

Load Distribution Configuration

```
# Cloud Run configuration
autoscaling.knative.dev/minScale: ${MIN_INSTANCES}
autoscaling.knative.dev/maxScale: ${MAX_INSTANCES}
```

```
containerConcurrency: 100
timeoutSeconds: 300
```

6.1.1.5 Circuit Breaker Patterns

Current Implementation Status

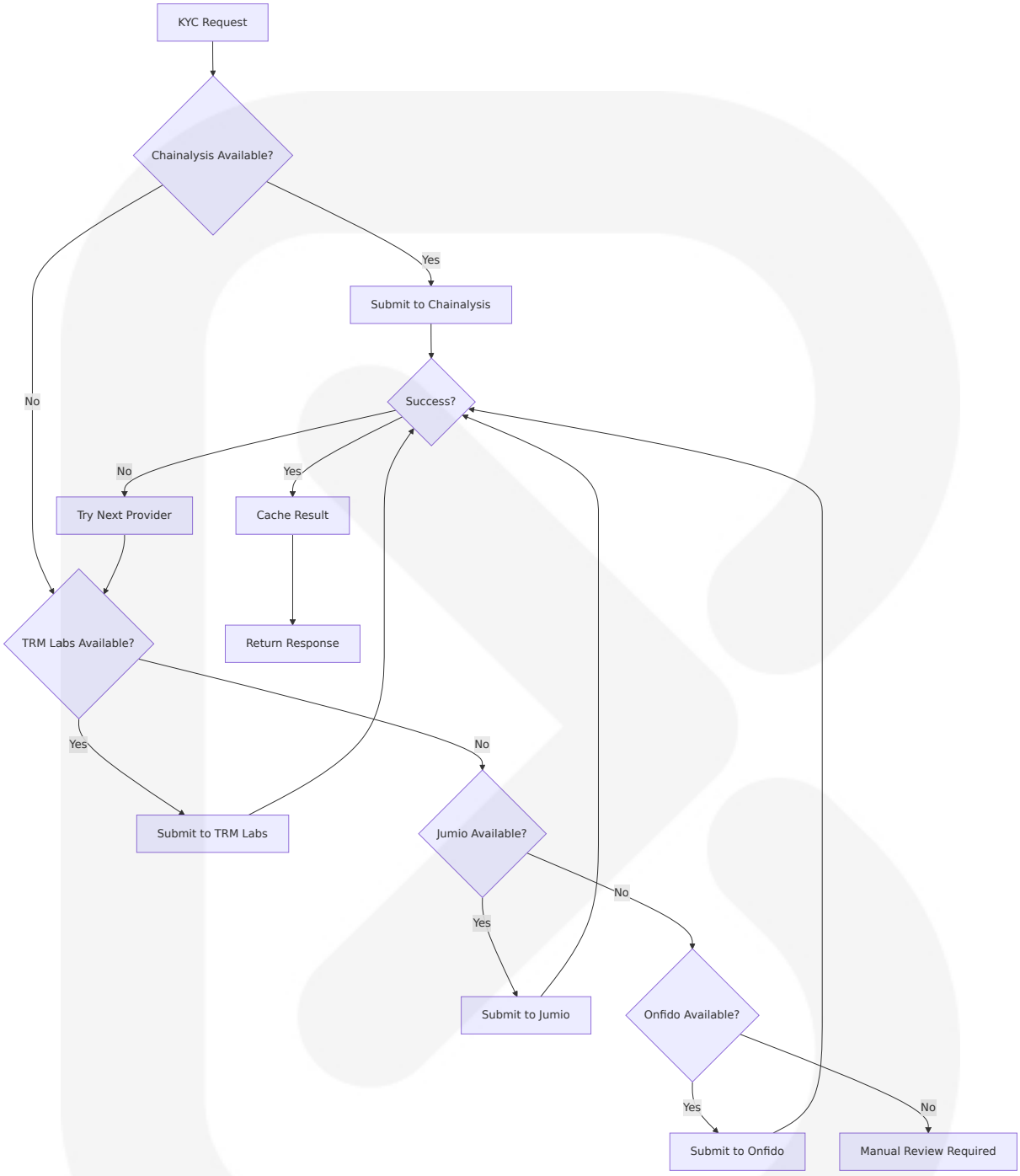
Circuit Breaker Implementation: Currently **not implemented** - identified as a gap in the resilience architecture. Services rely on timeout-based failure handling without circuit breaking logic.

Planned Implementation Areas:

- KYC provider service integration (high external dependency risk)
- Blockchain service contract interactions (network volatility)
- Database connection pooling (resource exhaustion protection)

Fallback Mechanisms

Multi-Provider Fallback: The KYC Provider service implements sophisticated fallback logic:



6.1.1.6 Retry and Fallback Mechanisms

Retry Strategy Implementation

Redis Connection Retry: Linear backoff strategy with maximum 2000ms delay for cache operations

Database Connection Retry: Connection pool exhaustion handling with automatic reconnection
External API Retry: Provider-specific retry logic with exponential backoff for KYC and blockchain operations

Graceful Degradation Patterns:

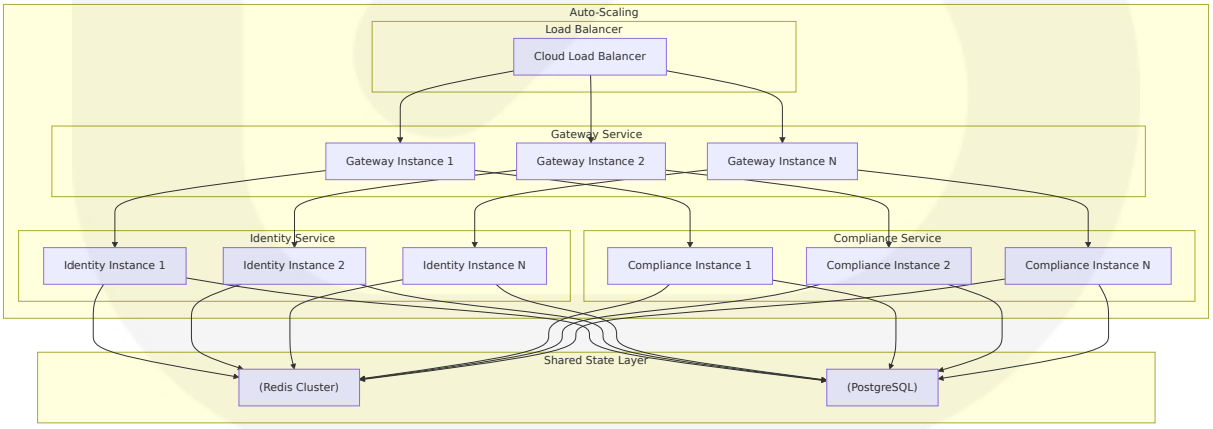
- **Cache Miss Tolerance**: Services continue operation without Redis, accepting performance degradation
- **Compliance Check Fallback**: Default to manual review when automated screening fails
- **Report Generation Fallback**: Multiple format options (PDF, CSV, JSON) with format-specific error handling

6.1.2 SCALABILITY DESIGN

6.1.2.1 Horizontal and Vertical Scaling Approach

Cloud-Native Horizontal Scaling

Container-Based Horizontal Scaling: All services implement stateless, horizontally scalable architectures through Cloud Run's serverless container platform:



Vertical Scaling Configuration

Resource Allocation Strategy:

- **Production CPU**: 0.5 to 2.0 CPU cores per container instance
- **Production Memory**: 512Mi to 2Gi RAM per container instance
- **Development Resources**: Constrained to single-core, 512Mi configurations for cost optimization
- **Storage**: Ephemeral container storage with persistent data in managed databases

6.1.2.2 Auto-Scaling Triggers and Rules

Cloud Run Auto-Scaling Configuration

Scaling Triggers:

- **CPU Utilization**: Target 70% CPU utilization across instances
- **Request Volume**: 100 concurrent requests per container before scaling
- **Memory Pressure**: 80% memory utilization threshold
- **Request Latency**: Scale up when p95 latency exceeds 500ms

Scaling Rules:

Metric	Threshold	Scale-Up Action	Scale-Down Delay
CPU Utilization	>70%	Add 1 instance	10 minutes
Concurrent Requests	>80 per instance	Add 1 instance	15 minutes
Memory Usage	>80%	Add 1 instance	5 minutes
Request Queue Depth	>10 requests	Add 2 instances	20 minutes

6.1.2.3 Resource Allocation Strategy

Service-Specific Resource Profiles

High-Compute Services:

- **Compliance Service**: 2 CPU cores, 2Gi RAM (complex rule evaluation)
- **Regulatory Reporting**: 1.5 CPU cores, 1.5Gi RAM (document generation)
- **KYC Provider**: 1 CPU core, 1Gi RAM (I/O bound external calls)

Standard Services:

- **Gateway Service**: 1 CPU core, 512Mi RAM (lightweight proxying)
- **Identity Service**: 0.5 CPU cores, 512Mi RAM (JWT token operations)
- **Policy Service**: 0.5 CPU cores, 512Mi RAM (policy evaluation)

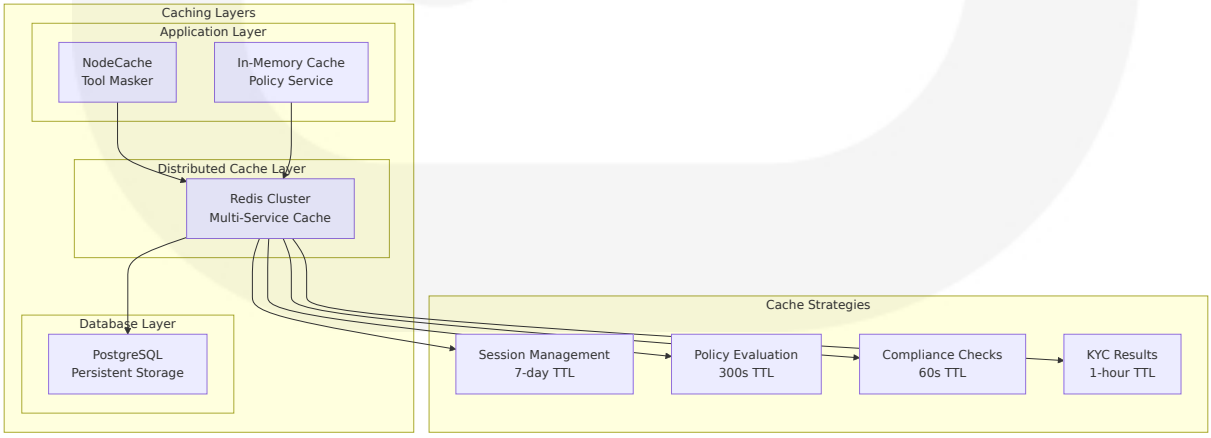
Database Connection Pool Scaling

PostgreSQL Connection Management:

- **Pool Size**: 20 connections per service instance
- **Max Overflow**: 10 additional connections during peak load
- **Idle Timeout**: 30 seconds for inactive connections
- **Connection Recycling**: 3600 seconds maximum connection lifetime

6.1.2.4 Performance Optimization Techniques

Multi-Tier Caching Architecture



Framework-Level Optimizations

Fastify Performance Features:

- **Schema Validation**: Pre-compiled JSON schemas for request/response validation
- **Serialization**: Fast JSON serialization with pre-defined schemas
- **Connection Keep-Alive**: HTTP/1.1 connection pooling for inter-service communication
- **Async/Await**: Non-blocking I/O operations throughout service implementations

6.1.2.5 Capacity Planning Guidelines

Service Instance Sizing Matrix

Service	Min Instances	Max Instances	Target Utilization	Scaling Factor
Gateway Service	2	50	70% CPU	2x per minute
Identity Service	1	20	70% CPU	1.5x per minute
Compliance Service	2	30	60% CPU	1.8x per minute
KYC Provider	1	15	80% CPU	1.2x per minute

Capacity Planning Metrics:

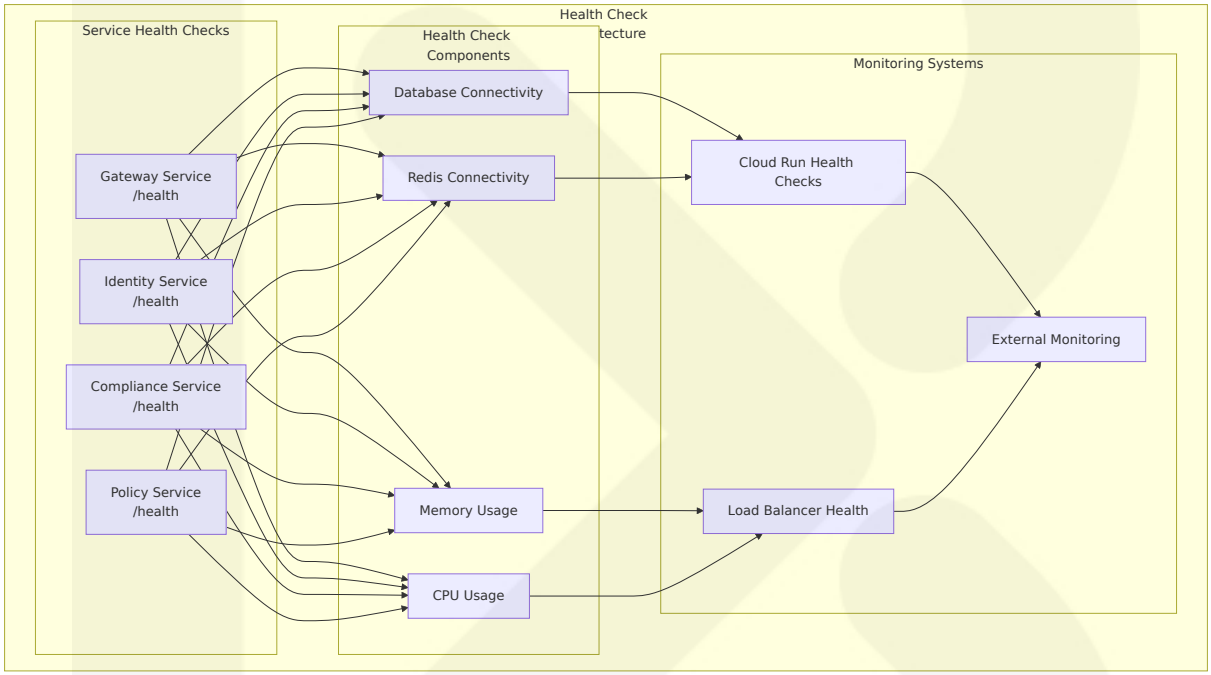
- **Peak Load Handling**: 10,000 TPS across all services
- **Response Time Targets**: <200ms p99 latency for API operations
- **Throughput Requirements**: 30,000 requests per second Gateway capacity
- **Concurrent User Support**: 1,000 simultaneous compliance officers

6.1.3 RESILIENCE PATTERNS

6.1.3.1 Fault Tolerance Mechanisms

Health Check Implementation

Every service implements comprehensive health monitoring through standardized `/health` endpoints:



Graceful Shutdown Patterns

Signal Handling Implementation: All services implement SIGTERM/SIGINT handlers for graceful resource cleanup:

1. **Connection Draining**: Stop accepting new requests while completing in-flight operations
2. **Database Pool Closure**: Graceful PostgreSQL connection pool shutdown
3. **Redis Disconnection**: Proper Redis client disconnection with pending operation completion

4. **Server Closure**: Fastify server graceful shutdown with timeout handling

Error Boundary Implementation

Centralized Error Handling: Services implement consistent error handling with:

- **Zod Schema Validation**: Runtime type checking with detailed error messages
- **HTTP Status Code Mapping**: Standardized error response formats across services
- **Request Context Logging**: Correlation ID tracking for distributed error diagnosis

6.1.3.2 Disaster Recovery Procedures

Data Backup and Recovery Strategy

Database Backup Configuration:

- **PostgreSQL Point-in-Time Recovery**: Continuous WAL archiving with 30-day retention
- **Daily Full Backups**: Automated nightly backups with geographic replication
- **Transaction Log Shipping**: Real-time backup of transaction logs for minimal data loss

Recovery Time Objectives (RTO):

- **Database Recovery**: <30 minutes for full database restoration
- **Service Recovery**: <5 minutes for individual service instance replacement
- **Complete System Recovery**: <60 minutes for full platform restoration

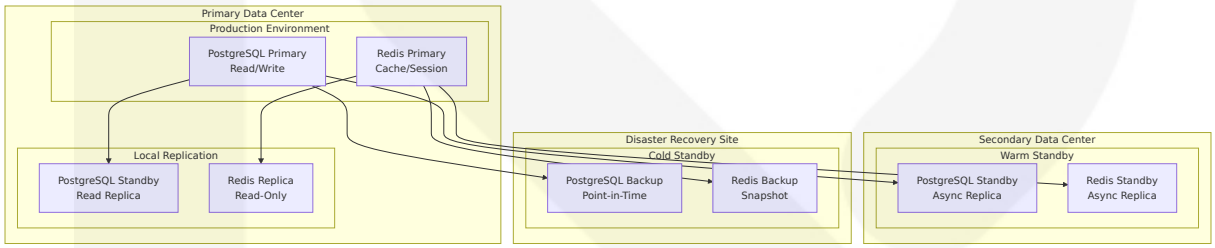
Cross-Region Resilience

Multi-Region Deployment Strategy:

- **Primary Region**: us-central1 (Google Cloud Run)
- **Secondary Region**: us-east1 (warm standby)
- **Disaster Recovery Region**: europe-west1 (cold standby)

6.1.3.3 Data Redundancy Approach

Database Replication Architecture



Audit Trail Redundancy

Dual-Write Pattern: The Audit Log Writer service implements dual-write persistence:

- **Primary Storage**: JSONL file format for immediate availability
- **Secondary Storage**: PostgreSQL for query capabilities and long-term retention
- **Integrity Verification**: Hash-based verification for tamper detection

6.1.3.4 Failover Configurations

Automatic Failover Mechanisms

Service-Level Failover: Cloud Run implements automatic instance replacement upon health check failure:

- **Health Check Frequency**: Every 10 seconds
- **Failure Threshold**: 3 consecutive failures trigger replacement

- **Replacement Time**: <60 seconds for new instance availability

Database Failover Configuration:

- **Primary-to-Standby Promotion**: <30 seconds automatic failover
- **Connection String Updates**: DNS-based failover with 5-second TTL
- **Application Reconnection**: Automatic connection pool recovery

KYC Provider Failover

Multi-Provider Resilience: Intelligent provider routing with health-based selection:

Provider	Primary Use Case	Fallback Priority	Availability SLA
Chainalysis	Sanctions screening	1st choice	99.9% uptime
TRM Labs	Risk assessment	2nd choice	99.5% uptime
Jumio	Document verification	3rd choice	99.0% uptime
Onfido	Biometric verification	4th choice	99.0% uptime

6.1.3.5 Service Degradation Policies

Graceful Degradation Strategies

Cache-Miss Tolerance: Services continue operation with degraded performance when Redis is unavailable:

- **Policy Evaluation**: Direct database queries with increased latency
- **Session Management**: Shorter-lived JWT tokens without Redis session tracking
- **Compliance Caching**: Real-time rule evaluation without result caching

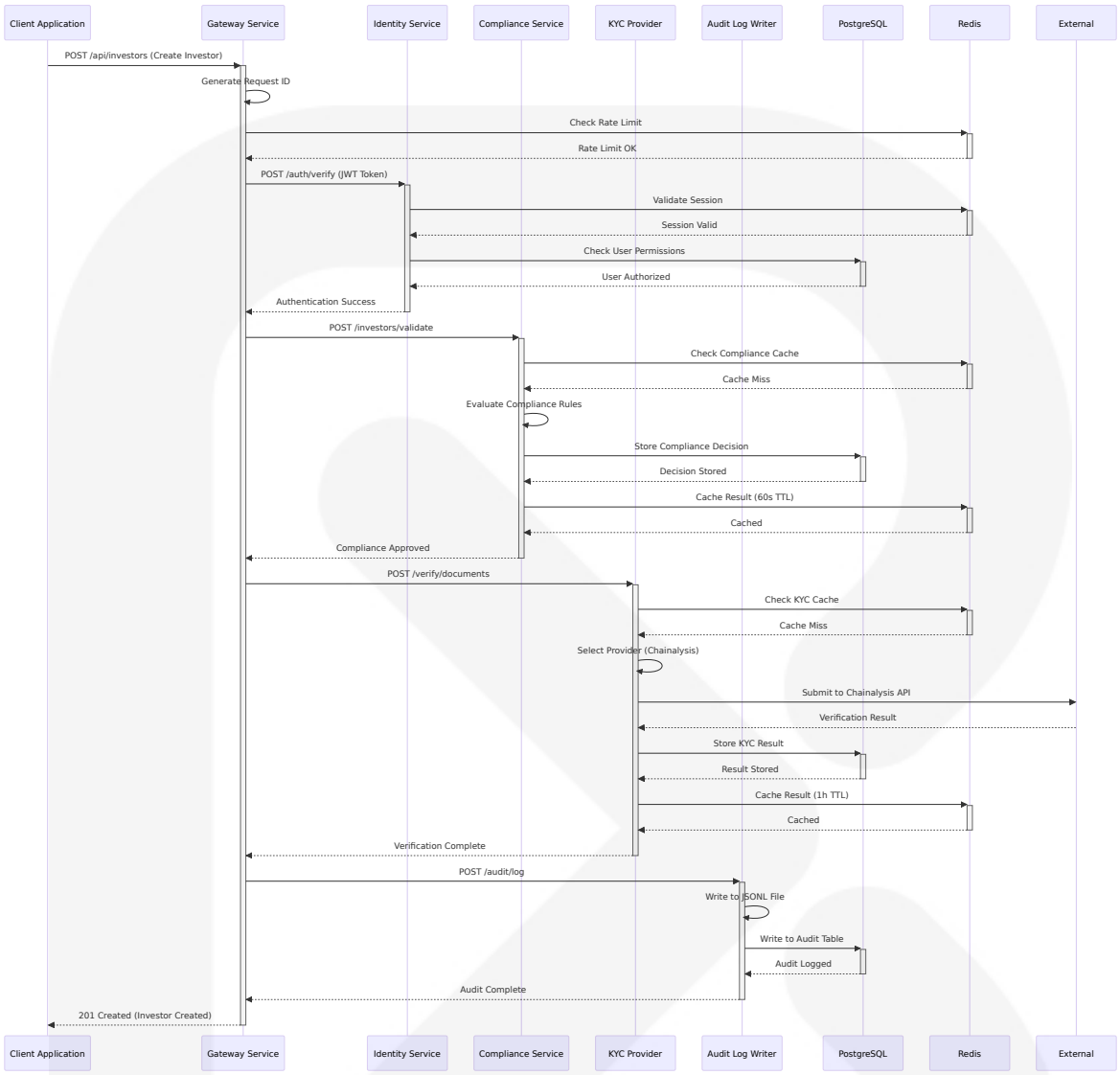
Feature Degradation Matrix:

Service	Full Operation	Degraded Mode	Emergency Mode
Gateway Service	All routes active	Rate limiting disabled	Read-only operations
Identity Service	JWT + sessions	JWT only	Manual authentication
Compliance Service	Automated decisions	Manual review required	Block all transactions
KYC Provider	All providers active	Single provider only	Manual verification

6.1.4 SERVICE INTERACTION PATTERNS

6.1.4.1 Request Flow Architecture

The following diagram illustrates the complete request flow through Veria's service mesh, highlighting authentication, authorization, and data flow patterns:



References

Repository Files Examined:

- `services/gateway/src/server.js` - Gateway routing implementation and proxy logic
- `services/gateway/src/config.js` - Service URL configuration and environment parsing
- `docker-compose.yml` - Local development orchestration and service dependencies
- `.env.example` - Environment configuration template with service ports

- `cloudrun.yaml` - Production deployment configuration for Google Cloud Run
- `Dockerfile` - Multi-stage build configuration for production containers

Repository Folders Explored:

- `` (root) - Repository structure analysis and monorepo configuration
- `services/` - Complete microservices architecture overview and service inventory
- `services/gateway/` - Gateway service implementation and routing patterns
- `services/compliance-service/` - Compliance service architecture and business logic
- `services/identity-service/` - Identity and authentication service implementation
- `services/kyc-provider/` - Multi-provider KYC integration patterns and fallback logic
- `packages/auth-middleware/` - Shared authentication and authorization middleware
- `packages/database/` - Database connection management and pooling configuration
- `infra/` - Infrastructure configuration and deployment automation
- `.github/` - CI/CD pipeline configuration and automated deployment workflows

Technical Specification Sections Referenced:

- `5.1 HIGH-LEVEL ARCHITECTURE` - System architecture context and design principles
- `3.2 FRAMEWORKS & LIBRARIES` - Technology stack and framework selection rationale
- `3.5 DATABASES & STORAGE` - Data persistence and caching strategy implementation
- `4.1 SYSTEM WORKFLOWS` - Service interaction patterns and business process flows

6.2 DATABASE DESIGN

6.2.1 DATABASE ARCHITECTURE OVERVIEW

6.2.1.1 Multi-Tier Database Strategy

The Veria compliance middleware platform implements a sophisticated multi-tier database architecture designed to handle the demanding requirements of financial compliance and regulatory technology. The architecture leverages three specialized database systems, each optimized for specific use cases within the compliance workflow:

Primary Relational Database: PostgreSQL v14 serves as the authoritative system of record for all transactional data, compliance records, and user management. The database provides ACID compliance essential for financial operations, advanced indexing capabilities for complex queries, and robust JSON support for flexible compliance rule storage.

Distributed Caching Layer: Redis v7 implements high-performance caching and session management with specialized configurations for compliance platform requirements. The system provides sub-millisecond response times for frequently accessed data and supports the platform's target of <200ms API response times.

Vector Search Database: Qdrant provides semantic search capabilities across compliance documentation and regulatory text, enabling intelligent document matching and compliance rule suggestion systems essential for automated compliance verification.

6.2.1.2 Database Selection Rationale

PostgreSQL Selection Criteria: PostgreSQL was selected over alternatives due to its exceptional performance characteristics for financial

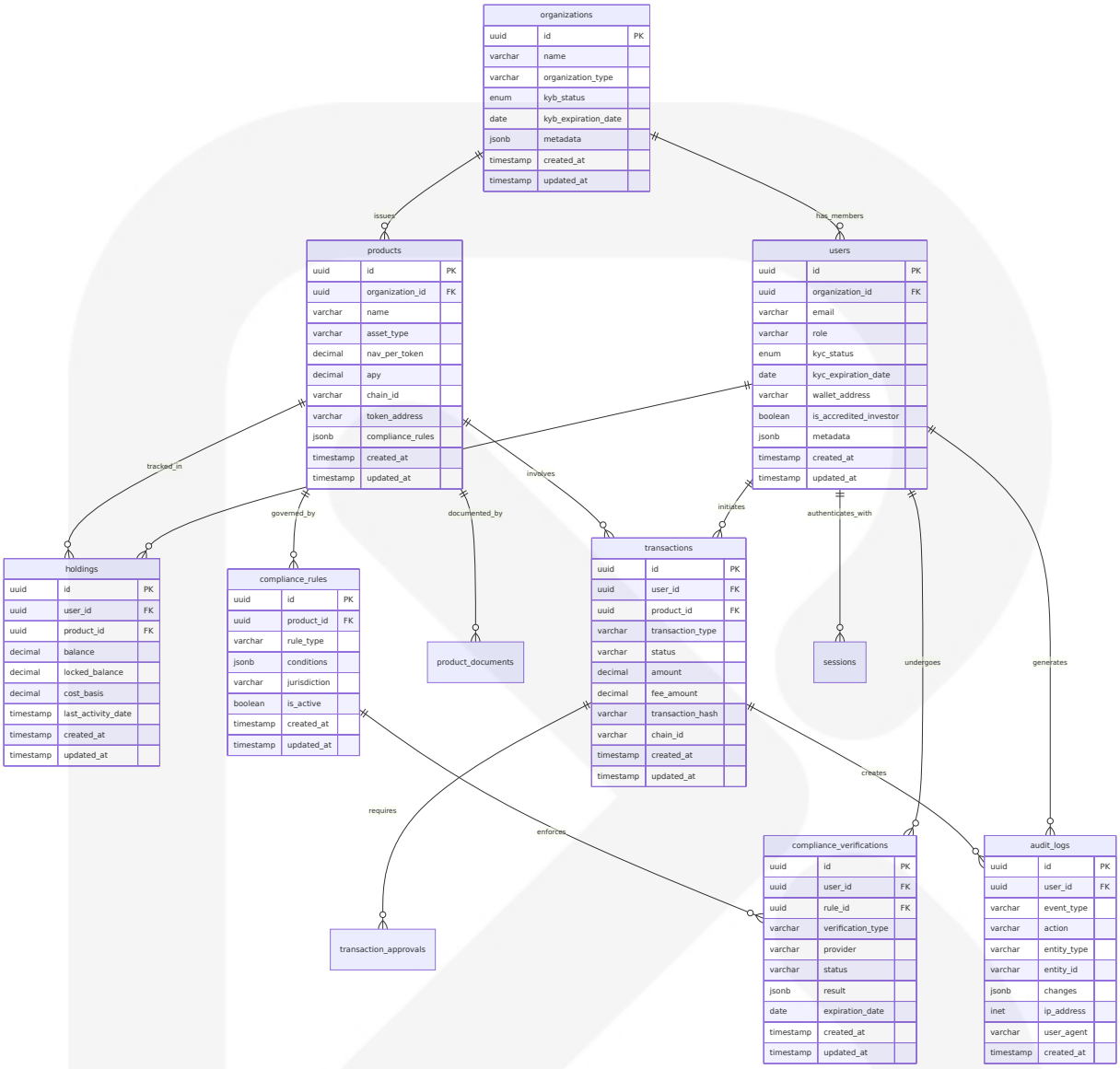
data processing, native JSON/JSONB support for flexible compliance rule storage, robust transaction isolation capabilities, and proven track record in financial services applications. The database supports the platform's requirement for 10,000+ concurrent connections through advanced connection pooling and provides the complex relational query capabilities needed for compliance reporting.

Redis Integration Strategy: Redis complements PostgreSQL by providing distributed session management with 7-day TTL for user sessions, rate limiting capabilities with 60-second windows, policy decision caching with 300-second TTL, and token blacklisting for security. This separation allows PostgreSQL to focus on persistent data operations while Redis handles ephemeral, high-frequency access patterns.

Vector Database Integration: Qdrant integration supports the platform's semantic search requirements for compliance documentation, enabling features such as regulatory text analysis, document similarity matching, and intelligent compliance rule suggestions. The system operates on both HTTP (port 6333) and gRPC (port 6334) protocols for optimal performance.

6.2.2 SCHEMA DESIGN

6.2.2.1 Entity Relationship Model



6.2.2.2 Core Domain Models

Organizations & Users Domain: The organization-centric design reflects the platform's B2B focus, supporting multiple organization types including issuers, distributors, investors, and service providers. Each organization maintains KYB (Know Your Business) status with expiration tracking, while users inherit organizational context and maintain individual KYC (Know Your Customer) status. The flexible JSONB metadata fields accommodate varying compliance requirements across different organization types and jurisdictions.

Products & Assets Domain: Product entities represent tokenized real-world assets with comprehensive financial tracking including NAV per token, APY calculations, and multi-chain deployment support. The schema supports the platform's initial focus on US Treasuries and Money Market Funds while providing flexibility for expansion to bonds, REITs, and commodities. Product documents maintain regulatory disclosure requirements with file hash verification and version control capabilities.

Compliance Domain: Dynamic compliance rules utilize JSONB storage for flexible rule conditions, supporting the platform's YAML-driven compliance engine. Verification records track KYC/KYB processes across multiple providers (Chainalysis, TRM Labs, Jumio, Onfido) with structured result storage and expiration management. This design enables real-time compliance decision-making while maintaining comprehensive audit trails.

Transaction Domain: Transaction entities support the complete lifecycle from subscription and redemption through dividend distribution and fee processing. Multi-level approval workflows are supported through the transaction_approvals table, enabling configurable authorization chains with role-based approval requirements. Holdings tracking provides real-time portfolio management with cost basis calculation for tax reporting.

6.2.2.3 Data Type Specifications

Financial Precision: All financial amounts utilize PostgreSQL's DECIMAL type with precision configurations optimized for specific use cases: DECIMAL(20, 8) for token amounts supporting fractional token holdings, DECIMAL(20, 2) for USD amounts maintaining standard currency precision, and DECIMAL(10, 4) for percentage values such as APY calculations.

Temporal Data Management: Comprehensive timestamp tracking utilizes PostgreSQL's native timestamp with time zone type, ensuring accurate temporal data across global operations. Expiration dates for KYC/KYB status use the date type for day-level precision, while transaction timing requires full timestamp precision for regulatory reporting.

Flexible Metadata Storage: JSONB columns provide structured flexibility for compliance rules, verification results, and extensible metadata. This approach enables rapid adaptation to regulatory changes while maintaining query performance through PostgreSQL's advanced JSONB indexing capabilities.

6.2.3 INDEXING STRATEGY

6.2.3.1 Primary and Foreign Key Indexes

UUID Primary Key Strategy: All tables utilize UUID primary keys generated via PostgreSQL's uuid-ossp extension, providing globally unique identifiers that prevent enumeration attacks while supporting distributed system operations. Primary key indexes utilize PostgreSQL's B-tree indexing for optimal performance across equality and range queries.

Foreign Key Performance: All foreign key relationships implement automatic indexing to optimize join operations essential for compliance reporting and audit queries. Composite foreign key indexes support multi-tenant data isolation ensuring organization-scoped data access patterns.

6.2.3.2 Business Logic Indexes

User Management Indexes: Email addresses maintain unique constraints with case-insensitive indexing using PostgreSQL's lower() function. Wallet addresses utilize unique constraints supporting blockchain integration requirements. Organization membership queries are optimized through composite indexes on (organization_id, role) combinations.

Compliance Query Optimization: KYC and KYB status fields utilize enum indexing for efficient filtering operations. Temporal indexes on expiration dates support automated compliance monitoring workflows. Rule evaluation queries benefit from composite indexes on (rule_type, jurisdiction, is_active) combinations.

Transaction Processing Indexes: Transaction status fields utilize partial indexes focusing on active transaction states (pending, processing) to optimize operational dashboards. Hash-based unique constraints on transaction_hash fields prevent double-spending while supporting cross-chain transaction tracking.

6.2.3.3 Audit and Reporting Indexes

Audit Trail Optimization: Entity reference indexes on (entity_type, entity_id) combinations enable efficient audit trail queries across any system entity. Temporal indexes on audit_logs.created_at support time-range audit queries essential for regulatory reporting. IP address indexes utilize PostgreSQL's INET type for network-based audit analysis.

Performance Monitoring: Query performance monitoring identifies slow-running queries for index optimization. Composite indexes on frequently joined tables reduce query execution times for complex compliance reports. Partial indexes on active records minimize index overhead while maximizing query performance.

6.2.4 PERFORMANCE OPTIMIZATION

6.2.4.1 Connection Pool Management

SQLAlchemy QueuePool Configuration: Database connection pooling utilizes SQLAlchemy's QueuePool with pool_size=20 base connections and max_overflow=10 additional connections, supporting the platform's requirement for 10,000+ concurrent users. Pool configuration includes pool_pre_ping=True for connection health validation and pool_recycle=3600 seconds to prevent stale connections.

Connection Health Management: Automatic connection recycling prevents long-lived connection issues while connection timeout configuration (2000ms) ensures responsive failure handling. The pool

implementation supports database maintenance operations through graceful connection draining.

6.2.4.2 Query Optimization Patterns

Complex Query Performance: Database queries target <500ms execution time for complex operations through strategic indexing and query optimization. Compliance rule evaluation queries utilize efficient JSON path operations with GIN indexes on JSONB columns. Reporting queries implement pagination patterns to manage large result sets.

Caching Integration: Query result caching through Redis reduces database load for frequently accessed data. Policy decision caching with 300-second TTL optimizes repeated compliance evaluations. KYC result caching with 1-hour TTL reduces external provider API calls while maintaining data freshness.

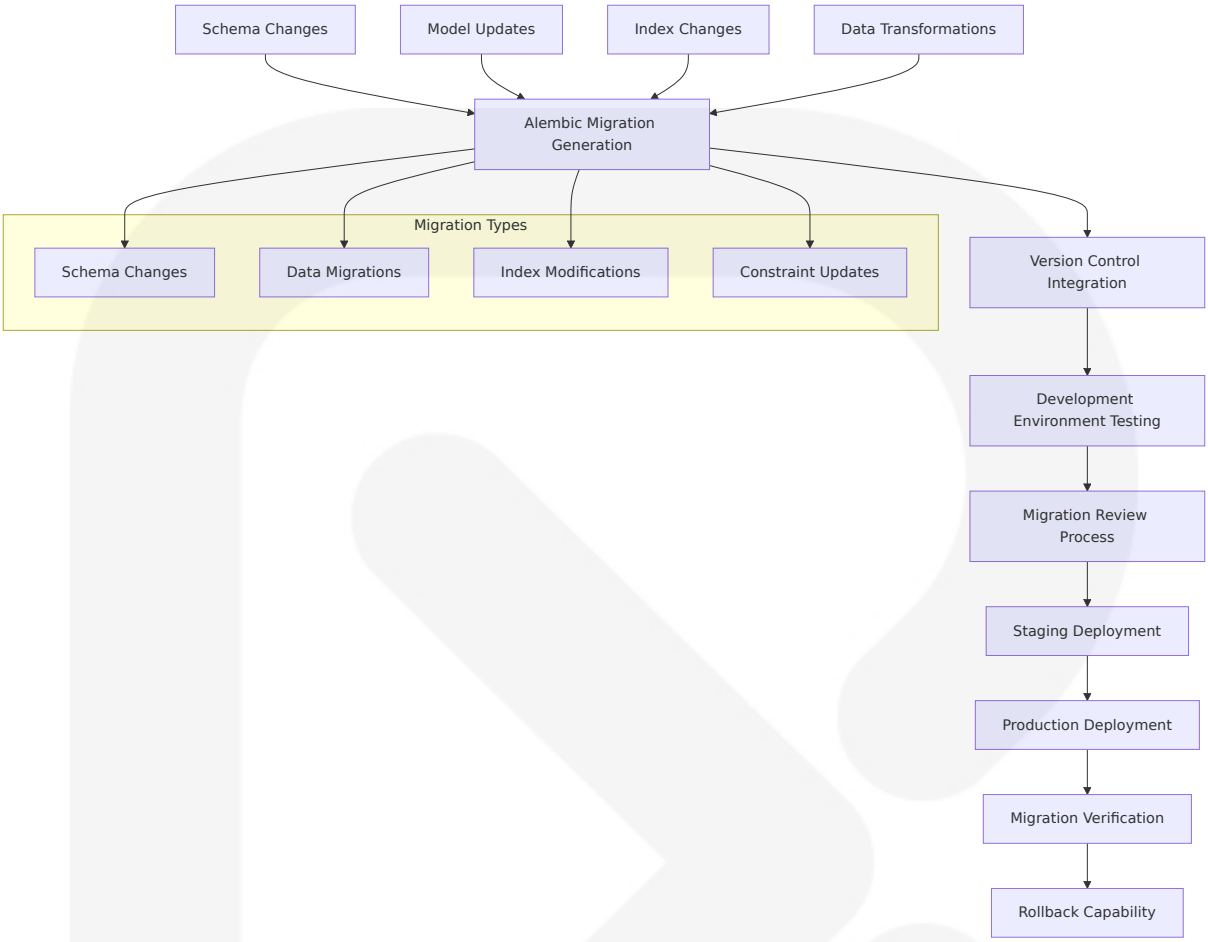
6.2.4.3 Read/Write Optimization

Transaction Isolation: Financial operations utilize PostgreSQL's transaction isolation capabilities to ensure data consistency during concurrent operations. Optimistic locking patterns prevent race conditions in balance updates while maintaining system performance.

Batch Processing Strategy: Large data operations utilize batch processing patterns to minimize database load. Report generation implements streaming query patterns for memory-efficient processing of large datasets. Migration operations utilize chunked processing to maintain system availability during schema updates.

6.2.5 DATA MANAGEMENT

6.2.5.1 Migration Framework



Alembic Migration System: Database schema evolution utilizes Alembic for SQLAlchemy-based migration management, providing version-controlled schema changes with automatic migration script generation. The system supports both schema modifications and data transformations while maintaining rollback capabilities for deployment safety.

Environment Management: Migration scripts support environment-specific configurations with separate database URLs for development (`veria_dev`) and production (`veria`) environments. Development seed data initialization through `init_db.py` provides consistent testing environments while maintaining production data integrity.

6.2.5.2 Data Initialization and Seeding

Core Schema Deployment: Initial database setup utilizes `packages/database/schemas/core.sql` for PostgreSQL DDL deployment,

ensuring consistent schema creation across environments. The initialization process includes extension activation (uuid-ossp) and core constraint configuration.

Development Data Seeding: Test data generation provides realistic datasets for development and testing environments using SQLite in-memory databases for test execution. Fixture management supports automated testing workflows while maintaining data consistency across test runs.

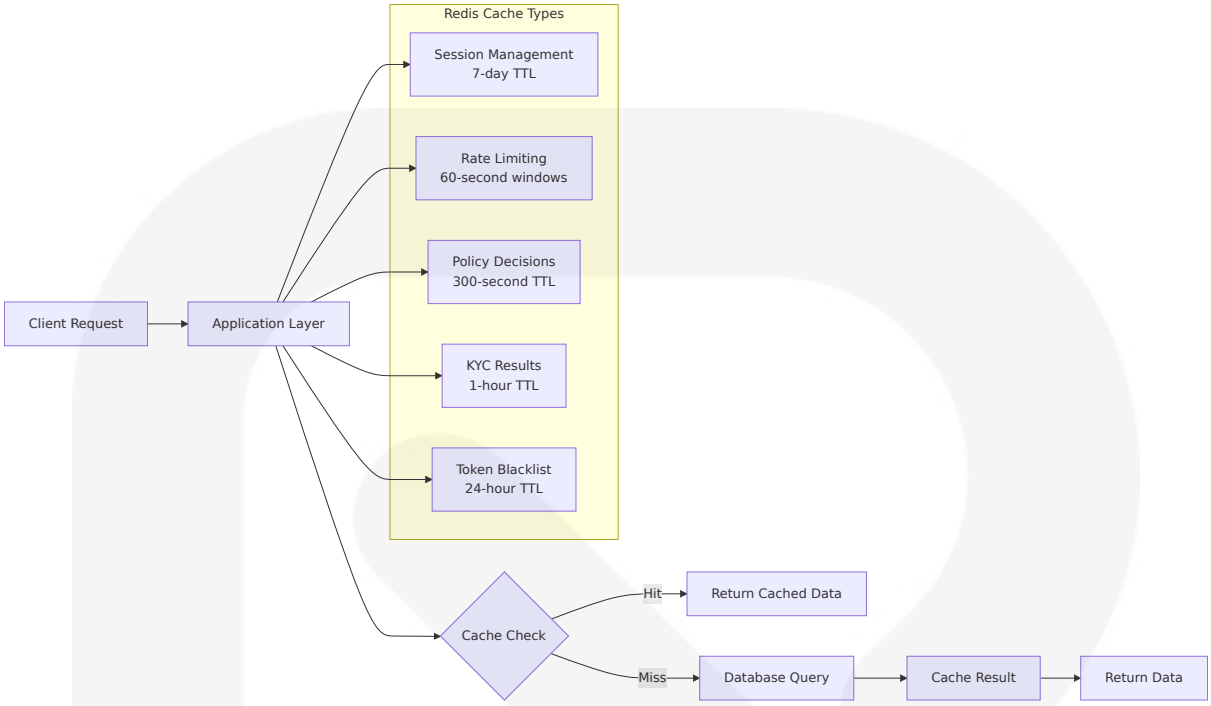
6.2.5.3 Versioning and Backup Strategy

Schema Version Control: Database schema versions align with application releases through semantic versioning. Migration scripts maintain forward and backward compatibility where possible, with explicit rollback procedures for breaking changes.

Data Retention Policies: Audit logs maintain permanent retention for regulatory compliance requirements. Transaction records retain permanent storage for financial reporting obligations. Session data implements 7-day retention with automatic cleanup. Cached verification results maintain 1-hour to 24-hour TTL based on data sensitivity.

6.2.6 CACHING ARCHITECTURE

6.2.6.1 Redis Caching Strategy



Session Management Caching: User sessions utilize Redis with 7-day TTL and sliding expiration, supporting distributed authentication across service instances. Session data includes JWT token hashes (not plaintext), user roles, and organization context for efficient authorization decisions.

Compliance Decision Caching: Policy evaluation results cache with 300-second TTL to optimize repeated compliance checks while maintaining reasonable data freshness for regulatory decision-making. KYC verification results cache with 1-hour TTL to reduce external provider API calls while ensuring compliance accuracy.

Security Token Management: JWT token blacklisting utilizes Redis with 24-hour TTL for revoked tokens, providing distributed token validation across service instances. Rate limiting implements 60-second windows with 100 request limits per user to prevent abuse.

6.2.6.2 Cache Invalidation Patterns

Time-Based Expiration: Automatic TTL-based expiration handles most cache invalidation scenarios with TTL values optimized for each data type's

staleness tolerance. Session data implements sliding expiration to maintain active user sessions while clearing inactive sessions.

Event-Driven Invalidation: Critical data changes trigger immediate cache invalidation through application-level cache management. Compliance rule updates invalidate policy decision caches to ensure immediate rule enforcement. User role changes invalidate session caches to enforce immediate permission changes.

6.2.7 COMPLIANCE CONSIDERATIONS

6.2.7.1 Regulatory Data Requirements

Immutable Audit Logging: Audit logs implement append-only patterns with immutable record design, ensuring complete transaction histories for regulatory compliance. The dual-write pattern ensures audit log persistence even during system failures, maintaining regulatory audit trail integrity.

Data Retention Compliance: Financial transaction data maintains permanent retention for regulatory reporting requirements. KYC/KYB verification records preserve complete verification histories with provider metadata for audit purposes. Compliance decision records maintain reasoning and rule version information for regulatory review.

6.2.7.2 Privacy and Access Controls

Data Isolation Architecture: Multi-tenant data isolation ensures organization-scoped data access through systematic foreign key relationships. User data access implements role-based authorization at the application layer with database-level organization boundaries.

PII Protection Strategy: Sensitive personal information utilizes application-level encryption before database storage. Database schema

design minimizes PII exposure through normalized entity relationships and metadata abstraction.

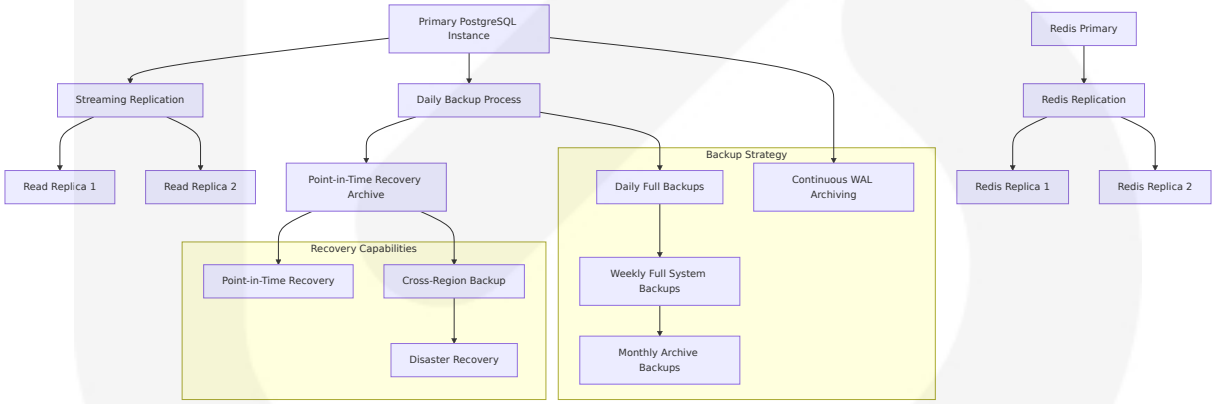
6.2.7.3 Audit Trail Implementation

Comprehensive Event Tracking: All significant system operations generate audit log entries with structured change tracking using JSONB columns. Event types include authentication, authorization, compliance decisions, transaction operations, and administrative actions.

Temporal Audit Capabilities: Audit logs maintain complete temporal sequences of system state changes, enabling point-in-time system state reconstruction for regulatory investigations. IP address tracking using PostgreSQL's INET type supports security audit requirements.

6.2.8 REPLICATION AND BACKUP ARCHITECTURE

6.2.8.1 High Availability Design



PostgreSQL Replication Strategy: Production deployment implements streaming replication with read replicas for query load distribution and high availability. Write operations target the primary instance while read operations distribute across replicas to optimize performance and provide failover capability.

Redis High Availability: Redis deployment utilizes master-replica configuration with automatic failover capabilities. Session data replication ensures continuous user experience during instance failures while maintaining distributed session state across service instances.

6.2.8.2 Backup and Recovery Procedures

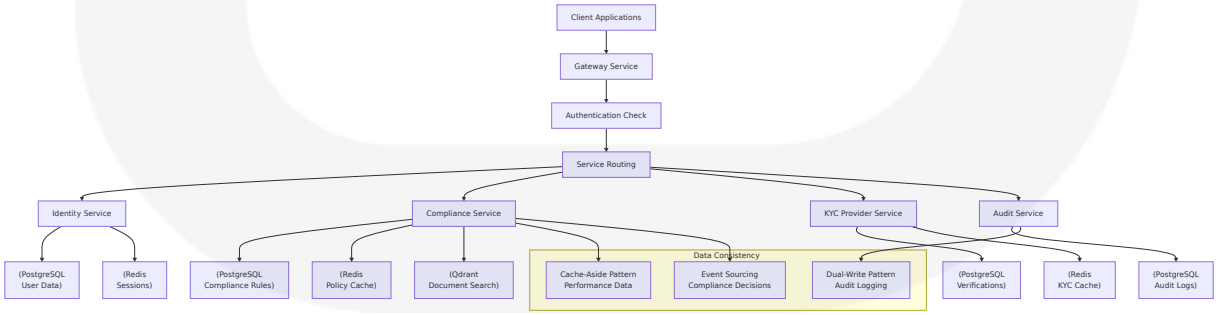
Continuous Backup Strategy: PostgreSQL implements continuous Write-Ahead Logging (WAL) archiving for point-in-time recovery capabilities essential for financial data protection. Daily full backups provide baseline recovery points while WAL archiving enables recovery to any point in time.

Cross-Environment Backup Management: Development environments utilize Docker volume persistence for local development while production implements cloud-native backup solutions. Backup verification procedures ensure recovery capability through regular restore testing.

Disaster Recovery Capabilities: Cross-region backup replication provides disaster recovery capabilities with defined Recovery Time Objectives (RTO) and Recovery Point Objectives (RPO) aligned with financial services requirements. Recovery procedures include full system restoration and partial data recovery scenarios.

6.2.9 DATA FLOW ARCHITECTURE

6.2.9.1 Core Data Flow Patterns



Primary Data Flow: Client requests flow through the Gateway service for authentication and routing to appropriate microservices. Each service

maintains dedicated database connections with service-specific data domains while sharing common audit logging patterns.

Compliance Decision Flow: Asset and investor data evaluation utilizes the Compliance Service with YAML-driven rule evaluation. Decision results implement caching patterns with Redis while maintaining permanent audit trails in PostgreSQL. Document similarity matching utilizes Qdrant for semantic search capabilities.

6.2.9.2 Integration Data Patterns

KYC Processing Flow: Multi-provider KYC verification implements orchestration patterns through the KYC Provider Service with fallback capabilities across providers (Chainalysis, TRM Labs, Jumio, Onfido). Results maintain structured storage in PostgreSQL with Redis caching for performance optimization.

Transaction Processing Flow: Financial transactions implement multi-stage processing with approval workflows, compliance verification, and blockchain interaction. Transaction state management utilizes database transactions for consistency while maintaining comprehensive audit trails.

6.2.10 VECTOR DATABASE INTEGRATION

6.2.10.1 Qdrant Semantic Search Architecture

Document Similarity Engine: Qdrant vector database supports semantic search across compliance documentation and regulatory text analysis. The system utilizes both HTTP (port 6333) and gRPC (port 6334) protocols for optimal performance based on operation type and data volume requirements.

Compliance Rule Matching: Vector similarity search enables intelligent compliance rule suggestions based on document content analysis and

regulatory context matching. This capability supports automated compliance verification and regulatory change impact analysis.

6.2.10.2 Search Integration Patterns

Hybrid Search Capabilities: Integration between PostgreSQL metadata search and Qdrant semantic search provides comprehensive document discovery capabilities. Query patterns combine structured metadata filtering with semantic content matching for optimal search results.

Performance Optimization: Vector search operations utilize efficient embedding strategies with cached vector representations for frequently accessed documents. Search result caching implements appropriate TTL values based on document change frequency and search pattern analysis.

References

Database Schema and Models

- `packages/database/models.py` - Complete SQLAlchemy ORM model definitions for all entities
- `packages/database/schemas/core.sql` - PostgreSQL DDL schema with table definitions and constraints

Database Configuration and Connection Management

- `packages/database/connection.py` - Database connection pooling and configuration
- `packages/database/.env.example` - Environment configuration template for database connections
- `docker-compose.yml` - Infrastructure configuration for PostgreSQL, Redis, and Qdrant

Migration and Data Management

- `packages/database/migrations/env.py` - Alembic migration configuration and environment setup
- `packages/database/init_db.py` - Database initialization and development seed data
- `packages/database/README.md` - Database package documentation and setup instructions

TypeScript Integration and Helpers

- `packages/database/src/index.ts` - TypeScript database helper functions and query templates
- `packages/database/tests/` - Database testing infrastructure and test configurations

Technical Specification Cross-References

- Section 3.5 DATABASES & STORAGE - Database technology selection and storage architecture
- Section 5.1 HIGH-LEVEL ARCHITECTURE - System architecture and data flow integration
- Section 2.5 NON-FUNCTIONAL REQUIREMENTS - Performance requirements affecting database design
- Section 2.1 FEATURE CATALOG - Feature requirements driving database schema design

6.3 INTEGRATION ARCHITECTURE

6.3.1 API DESIGN

6.3.1.1 Protocol Specifications

Veria implements a **RESTful API architecture** with JSON-based communication protocols, designed for high-performance financial

compliance operations. The system utilizes HTTP/1.1 with persistent connections and standardized content types across all service interfaces.

Core Protocol Standards:

- **Transport Protocol**: HTTPS/TLS 1.3 for all external communications
- **Content Type**: `application/json` for all request/response payloads
- **Character Encoding**: UTF-8 for all text-based communications
- **API Versioning**: URI path versioning (e.g., `/api/v1/investors`)

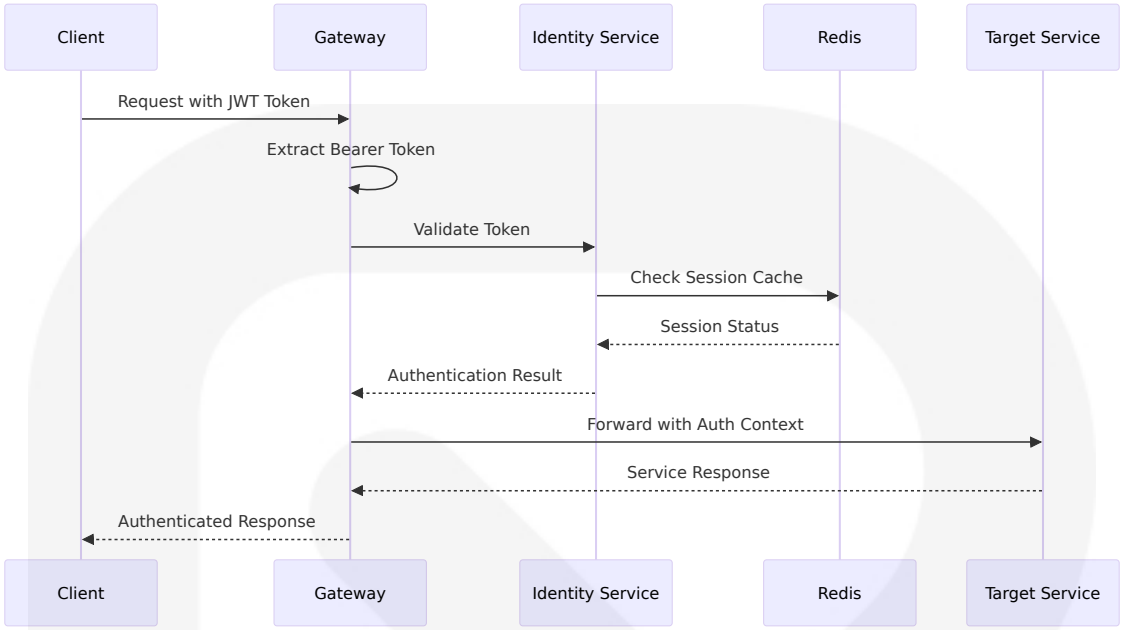
The Gateway service operates as the central API orchestrator on port 4000, implementing request proxying with header propagation and response transformation. All backend services (ports 4001-4005) communicate exclusively through this gateway, ensuring consistent protocol handling and centralized traffic management.

6.3.1.2 Authentication Methods

Primary Authentication Framework:
The system implements JWT-based authentication with Bearer token authorization, managed through the Identity Service and propagated via the Gateway's authentication middleware.

Authentication Method	Use Case	Token Lifetime	Implementation
JWT Bearer Tokens	User authentication	7 days	Auth0/JWT standard
API Key Authentication	Service-to-service	Permanent	Custom header validation
WebAuthn Support	Enhanced user security	Session-based	FIDO2 standard
Session Management	Web application state	7 days TTL	Redis-backed sessions

Token Propagation Flow:



6.3.1.3 Authorization Framework

Role-Based Access Control (RBAC) Implementation:

The authorization framework implements fine-grained permissions through role-based access control, with authorization decisions evaluated at both the Gateway and service levels.

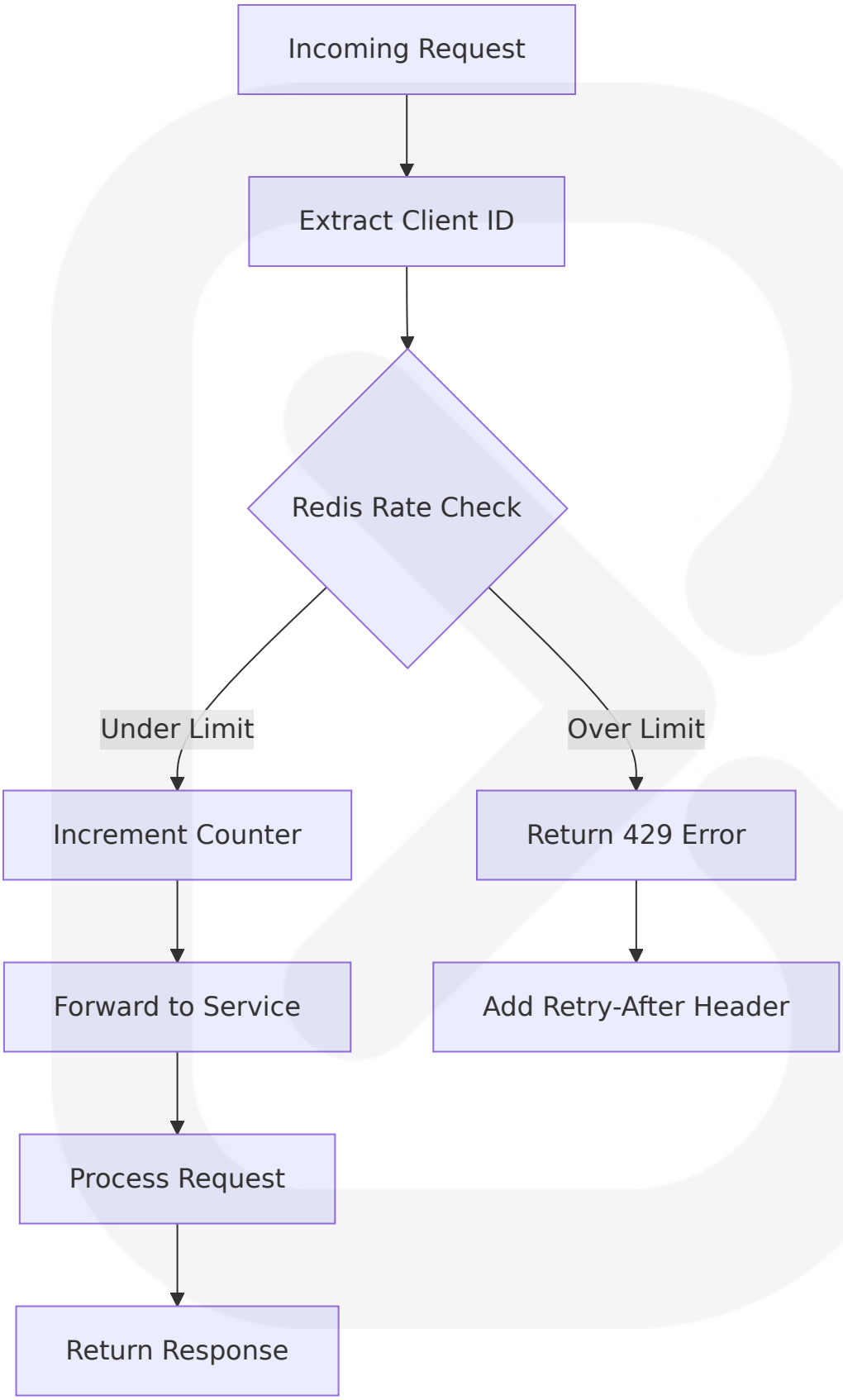
Authorization Levels:

Role	Access Scope	Permissions	Service Access
Compliance Officer	Full compliance operations	Read/Write all compliance data	All services
Investor	Personal data only	Read own records, submit documents	Identity, KYC services
Auditor	Read-only access	Audit trail queries, report generation	Audit, Reporting services
System Admin	System configuration	Service management, user administration	All services

6.3.1.4 Rate Limiting Strategy

Redis-Backed Rate Limiting:

The Gateway service implements distributed rate limiting using Redis counters with sliding window algorithms, ensuring consistent throttling across multiple service instances.


Incoming Request
Extract Client ID
Redis Rate Check
Under Limit
Over Limit
Increment Counter
Return 429 Error
Forward to Service
Add Retry-After Header
Process Request
Return Response

Rate Limiting Configuration:

Client Type	Requests/Minute	Burst Allowance	Reset Window
Authenticated Users	100	20 additional	60 seconds
API Clients	1000	100 additional	60 seconds
Public Endpoints	20	5 additional	60 seconds

6.3.1.5 Versioning Approach

URI Path Versioning Strategy:
Veria implements explicit version control through URI path segments, enabling backward compatibility and controlled API evolution without breaking existing integrations.

Version Management:

- **Current Version**: `v1` (initial production release)
- **Version Format**: `/api/v{major}/resource`
- **Deprecation Policy**: 12-month overlap for version transitions
- **Breaking Changes**: Require major version increment

6.3.1.6 Documentation Standards

OpenAPI Specification Compliance:
All APIs are documented using OpenAPI 3.1 specifications with comprehensive schema definitions, ensuring consistent documentation and enabling automated client generation.

Documentation Components:

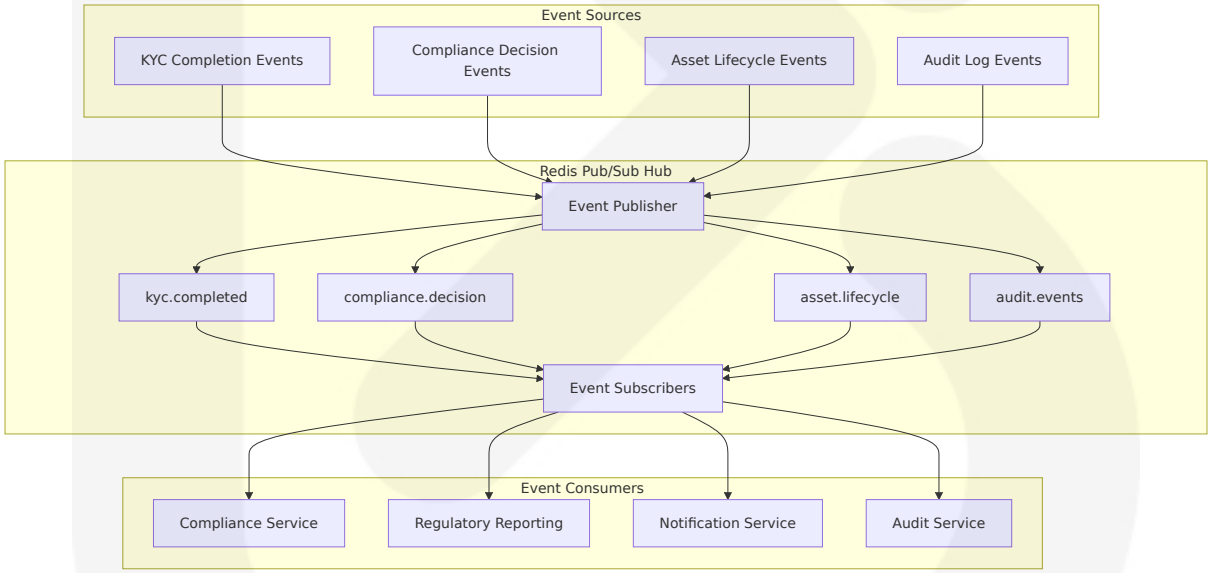
- **Endpoint Descriptions**: Detailed operation summaries with business context

- **Schema Validation**: Zod-powered runtime validation with OpenAPI export
- **Error Code Mapping**: Standardized HTTP status codes with detailed error messages
- **Authentication Requirements**: Per-endpoint security requirements documentation

6.3.2 MESSAGE PROCESSING

6.3.2.1 Event Processing Patterns

Redis Pub/Sub Architecture:
The system implements event-driven communication patterns using Redis pub/sub channels for asynchronous processing and real-time notifications across services.



Event Processing Characteristics:

- **Asynchronous Processing**: Non-blocking event propagation prevents service coupling
- **Event Persistence**: Critical events stored in PostgreSQL for replay capability

- **Channel Isolation**: Domain-specific channels prevent cross-contamination
- **Error Recovery**: Failed event processing triggers retry mechanisms

6.3.2.2 Message Queue Architecture

Queue-Based Processing Patterns:
While Redis pub/sub handles real-time events, the system implements queue-based patterns for reliable background processing and compliance workflows.

Processing Queues:

Queue Name	Purpose	Processing Pattern	Retry Strategy
compliance-screening	KYC document processing	Worker pool	3 attempts, exp backoff
regulatory-reports	Scheduled report generation	Batch processing	5 attempts, linear backoff
audit-processing	Audit log consolidation	Sequential processing	2 attempts, immediate
notification-delivery	User notifications	Concurrent processing	3 attempts, exp backoff

6.3.2.3 Stream Processing Design

Event Stream Architecture:
The system processes compliance events through stream-like patterns for real-time monitoring and automated decision-making.

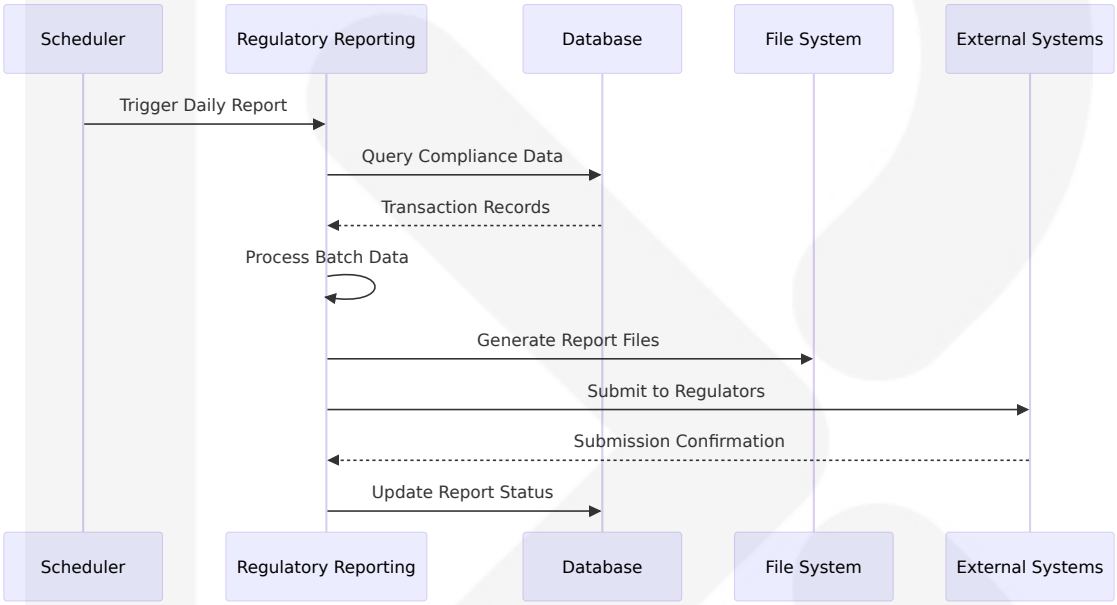
Stream Processing Components:

- **Event Ingestion**: Redis streams capture sequential event data
- **Window Operations**: Time-based aggregations for compliance metrics
- **Real-time Analytics**: Continuous monitoring of compliance violations

- **Stream Checkpointing**: Consumer group management for reliable processing

6.3.2.4 Batch Processing Flows

Regulatory Reporting Pipeline:
Large-scale data processing for regulatory compliance reports utilizes batch processing patterns with scheduled execution and failure recovery.



6.3.2.5 Error Handling Strategy

Multi-Tier Error Recovery:
The message processing system implements comprehensive error handling with automatic retry, dead letter queues, and manual intervention capabilities.

Error Handling Tiers:

1. **Immediate Retry**: Transient errors with exponential backoff
2. **Dead Letter Queues**: Failed messages for manual analysis
3. **Alert Generation**: Critical failures trigger operations notifications
4. **Manual Recovery**: Administrative tools for message reprocessing

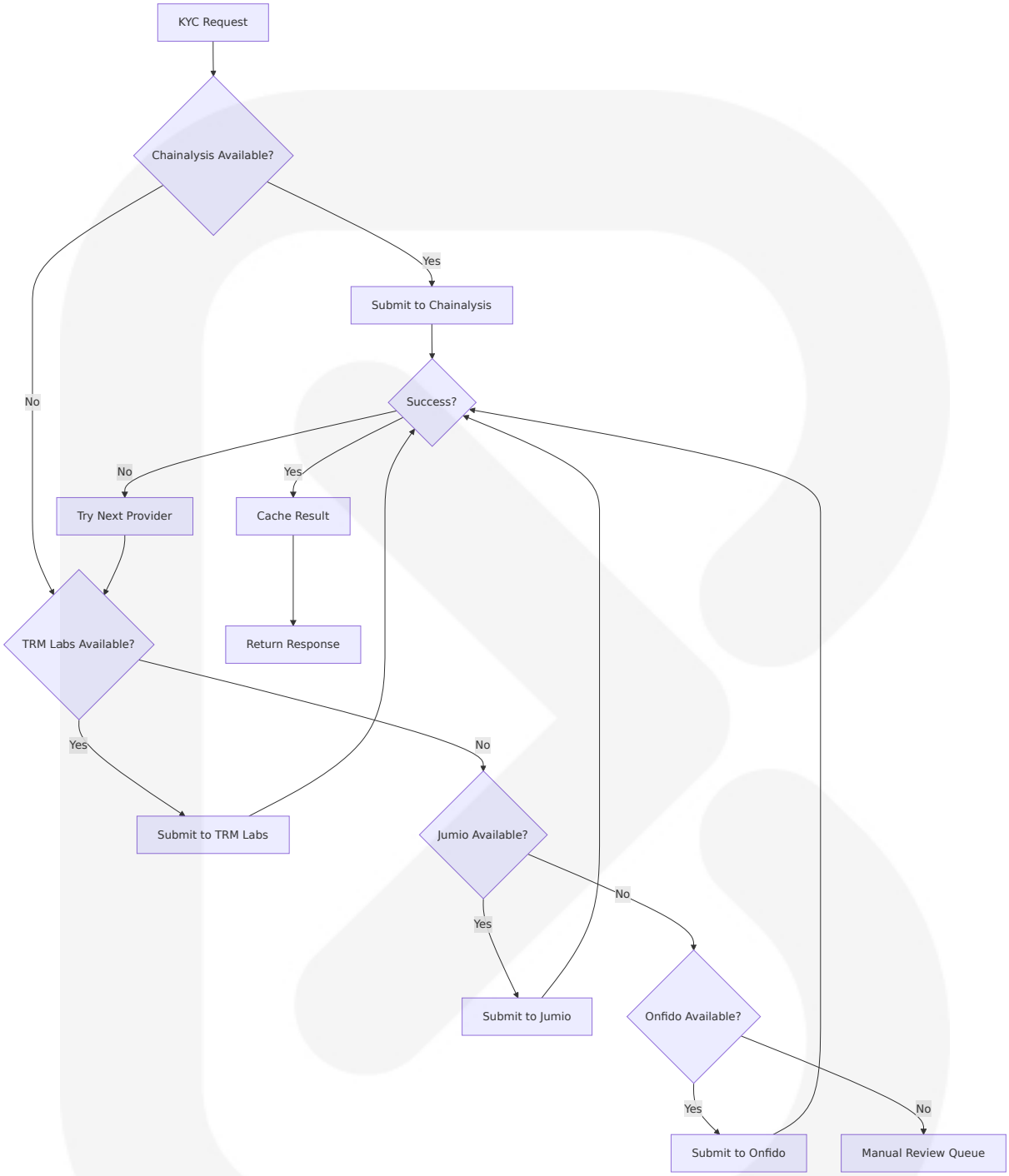
6.3.3 EXTERNAL SYSTEMS

6.3.3.1 Third-Party Integration Patterns

Multi-Provider Architecture:
Veria implements resilient external integrations with automatic failover mechanisms, ensuring system availability despite third-party service outages.

6.3.3.2 KYC Provider Ecosystem

Multi-Provider Failover Implementation:



Provider Integration Specifications:

Provider	Service Type	Response Time SLA	Availability SLA	Fallback Priority
Chainalysis	Blockchain screening	<5 seconds	99.9%	Primary

Provider	Service Type	Response Time SLA	Availability SLA	Fallback Priority
TRM Labs	Risk assessment	<3 seconds	99.5%	Secondary
Jumio	Document verification	<10 seconds	99.0%	Tertiary
Onfido	Biometric verification	<8 seconds	99.0%	Quaternary

6.3.3.3 Blockchain Network Integration

Multi-Chain Architecture:
The Blockchain Service integrates with multiple networks through standardized JSON-RPC interfaces, providing unified access to various blockchain ecosystems.

Supported Networks:

- **Ethereum Mainnet** (Chain ID: 1) - Primary deployment target
- **Polygon Network** (Chain ID: 137) - Production scaling solution
- **Solana Mainnet** - Alternative blockchain architecture
- **Testnets** - Development and testing environments

Integration Characteristics:

- **Provider Redundancy**: Alchemy primary, with Infura failover
- **Connection Pooling**: Persistent WebSocket connections for events
- **Rate Limiting**: Provider-specific request throttling
- **Error Recovery**: Automatic retry with exponential backoff

6.3.3.4 Custody Provider Integration

BNY Mellon Integration:
Traditional financial institution integration for asset custody services, implementing bank-grade security and compliance protocols.

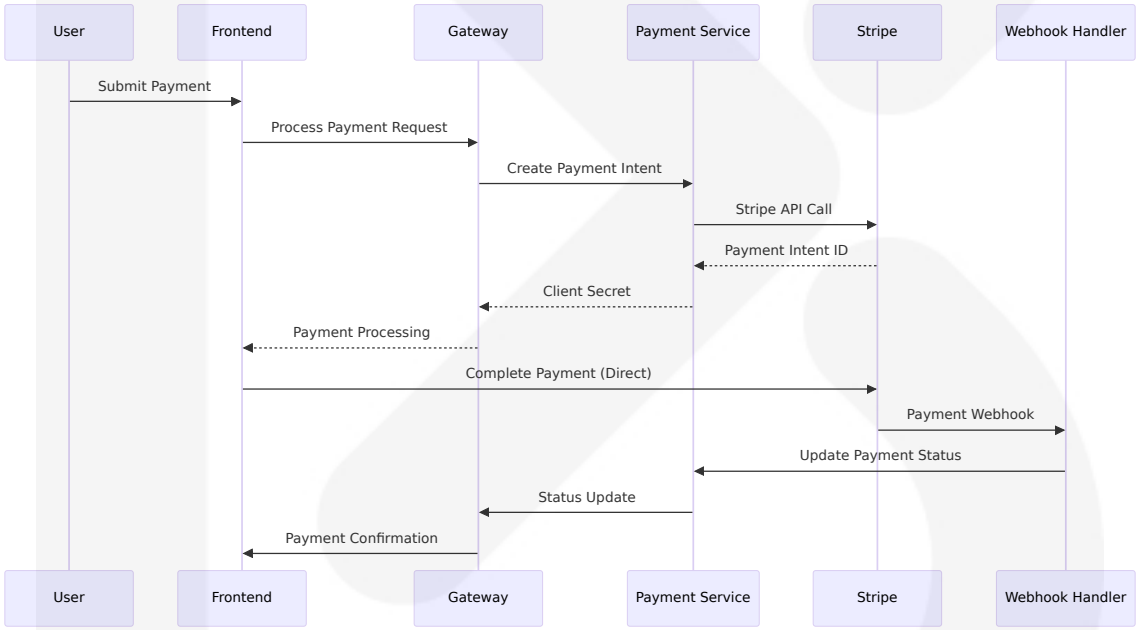
Integration Specifications:

- **Protocol**: HTTPS REST APIs with mutual TLS authentication
- **Data Format**: JSON with ISO 20022 financial messaging standards
- **Security**: API key authentication with request signing
- **Monitoring**: Real-time custody balance reconciliation

6.3.3.5 Payment Processing Integration

Stripe Integration Architecture:
PCI DSS compliant payment processing with webhook-based status updates and comprehensive fraud prevention.

Payment Flow Integration:



6.3.3.6 API Gateway Configuration

Gateway Service Architecture:
The Gateway service implements comprehensive API management with request routing, authentication propagation, and response transformation.

Gateway Configuration:

Feature	Implementation	Configuration
Request Routing	Fastify with route proxying	Dynamic service discovery
Load Balancing	Round-robin with health checks	Service instance tracking
Request Transformation	Header injection and validation	Middleware pipeline
Response Caching	Redis-backed response cache	TTL-based invalidation

6.3.3.7 External Service Contracts

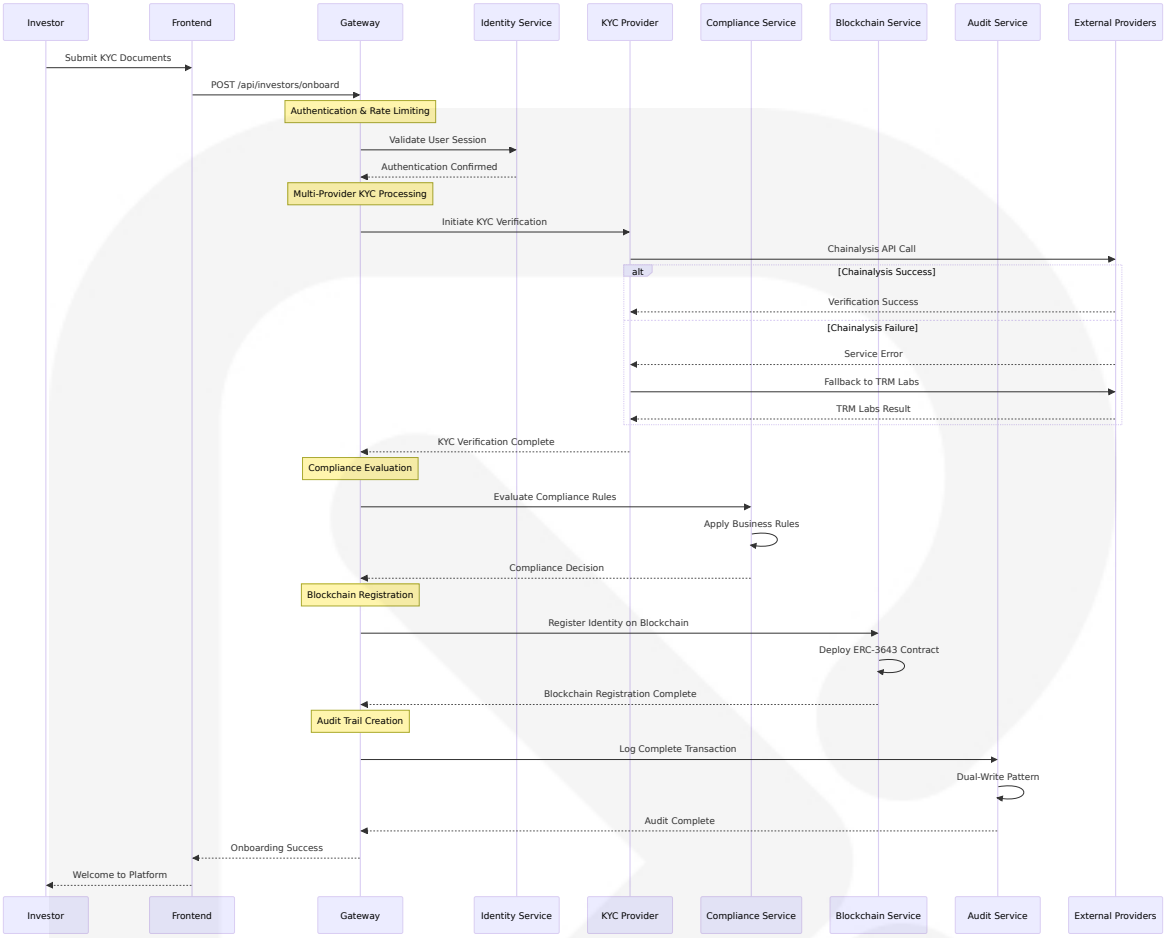
Service Level Agreements:

Service Category	Availability SLA	Response Time SLA	Error Rate SLA
Identity Providers	99.9% uptime	<2 seconds	<0.1% error rate
Payment Processors	99.95% uptime	<3 seconds	<0.05% error rate
Blockchain Networks	Network dependent	<10 seconds	<1% error rate
Custody Providers	99.99% uptime	<5 seconds	<0.01% error rate

6.3.4 INTEGRATION FLOW DIAGRAMS

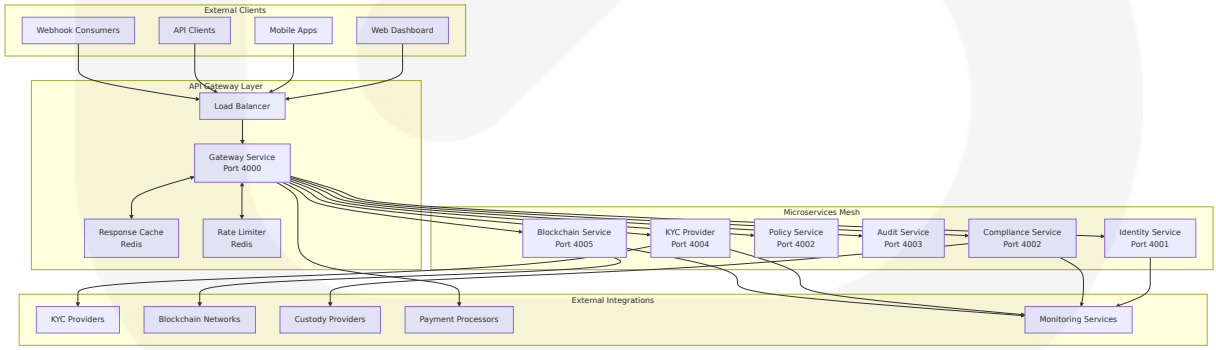
6.3.4.1 Complete Integration Flow

End-to-End Investor Onboarding Integration:



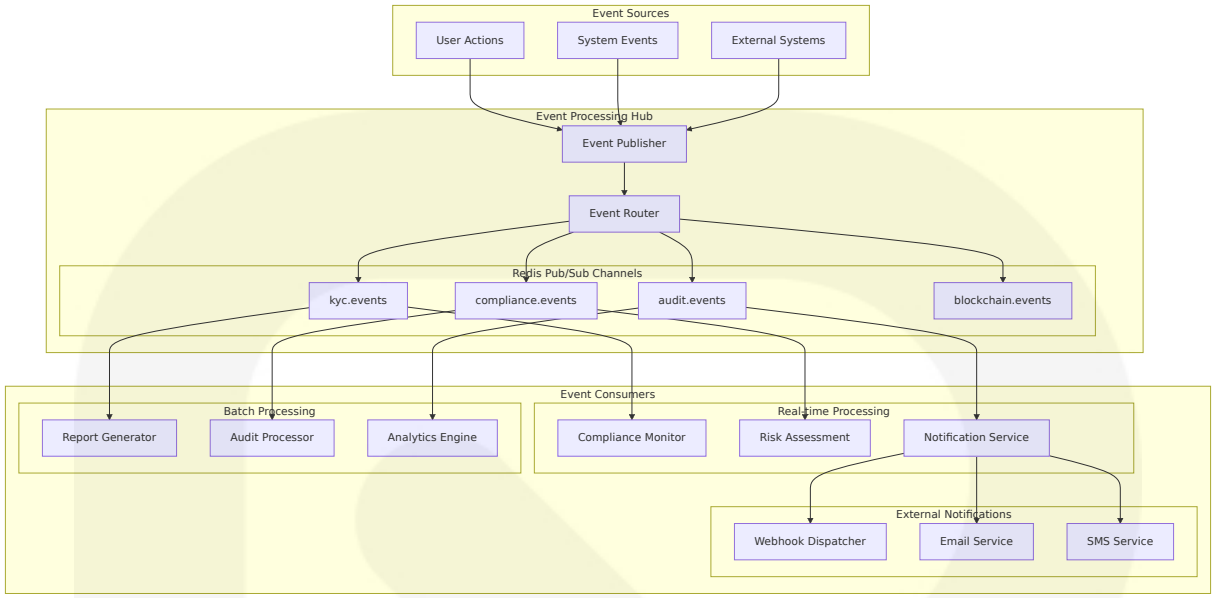
6.3.4.2 API Gateway Integration Architecture

Service Mesh Integration Pattern:



6.3.4.3 Message Flow Architecture

Event-Driven Integration Patterns:



References

Repository Files Examined:

- `services/gateway/src/server.js` - Gateway routing and proxy implementation with authentication middleware
- `services/gateway/src/config.js` - Service URL configuration and environment variable mapping
- `services/kyc-provider/src/manager.ts` - Multi-provider KYC orchestration with fallback mechanisms
- `docker-compose.yml` - Local development service orchestration and port configuration
- `.env.example` - Environment configuration template with external service API keys
- `cloudrun.yaml` - Production deployment configuration for Google Cloud Run

Repository Folders Explored:

- `` (root) - Monorepo structure analysis and integration architecture overview

- `services/` - Complete microservices architecture with integration patterns
- `services/gateway/` - API Gateway implementation with routing and middleware
- `services/gateway/src` - Gateway source code including proxy logic and configuration
- `services/kyc-provider/` - KYC provider service with multi-provider integration
- `services/kyc-provider/src` - KYC source code including provider management
- `services/kyc-provider/src/providers` - Individual provider implementations and API wrappers
- `services/blockchain-service/` - Blockchain integration service architecture
- `services/blockchain-service/src` - Blockchain service source code and network abstractions
- `services/compliance-service/` - Compliance service with external screening integration
- `services/regulatory-reporting/` - Regulatory reporting service with batch processing
- `packages/` - Shared packages overview including authentication middleware
- `packages/auth-middleware/` - Shared authentication and authorization components

Technical Specification Sections Referenced:

- `5.1 HIGH-LEVEL ARCHITECTURE` - System architecture context and integration principles
- `6.1 CORE SERVICES ARCHITECTURE` - Service communication patterns and mesh architecture
- `3.4 THIRD-PARTY SERVICES` - External service configurations and provider specifications

- `4.4 INTEGRATION SEQUENCE DIAGRAMS` - Existing integration flow documentation
- `4.3 PERFORMANCE AND MONITORING FLOWS` - Monitoring and observability integration patterns

6.4 SECURITY ARCHITECTURE

6.4.1 AUTHENTICATION FRAMEWORK

6.4.1.1 Identity Management System

Veria implements a robust identity management system through the **Identity Service** (`services/identity-service/src/auth/`), providing centralized authentication and user lifecycle management for the compliance middleware platform. The system manages user identities across multiple organizational contexts, supporting the complex requirements of tokenized asset compliance.

User Identity Model: The system maintains user identities with comprehensive profile information including organizational affiliations, role assignments, and compliance status. User records include authentication credentials, multi-factor authentication settings, and session tracking capabilities.

Organizational Context Management: The identity system supports multi-tenant architecture where users can have different roles and permissions across different organizations and asset classes, essential for compliance officers managing multiple funds or institutional relationships.

6.4.1.2 Multi-Factor Authentication

WebAuthn/FIDO2 Implementation: The platform implements passwordless authentication using the WebAuthn standard (`services/identity-service/src/auth/webauthn.ts`) with the

@simplewebauthn/server library. This provides enterprise-grade security suitable for financial compliance environments.

WebAuthn Configuration	Value	Purpose
Supported Algorithms	ES256 (-7), RS256 (-257)	Cryptographic signature verification
Challenge Expiry	5 minutes	Prevent replay attacks
User Verification	Required	Biometric or PIN confirmation
Resident Key	Required	Enable passwordless authentication

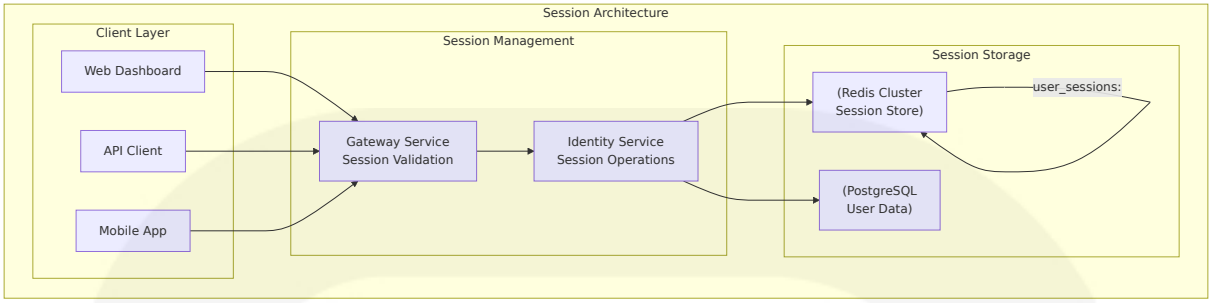
Authenticator Preferences: The system prioritizes platform authenticators (built-in biometric sensors) while supporting cross-platform authenticators (USB security keys) for maximum security coverage across different device types.

Relying Party Configuration: The WebAuthn implementation uses configurable relying party settings via environment variables:

- `WEBAUTHN_RP_NAME` : Default "Veria Platform"
- `WEBAUTHN_RP_ID` : Configurable domain identifier
- `WEBAUTHN_ORIGIN` : Origin validation for authentication requests

6.4.1.3 Session Management

Redis-Backed Distributed Sessions: The platform implements distributed session management using Redis (`services/identity-service/src/auth/session.ts`) with a 7-day time-to-live (TTL) and sliding expiration that refreshes on access.



Session Data Structure: Each session includes comprehensive metadata:

- User identification (userId, email, roles)
- Organizational context (organizationId)
- Security tracking (createdAt, lastAccessedAt, expiresAt)
- Request context (ipAddress, userAgent)

Session Lifecycle Management: The system provides automatic session refresh, bulk session invalidation for security incidents, and user-initiated logout across all devices through Redis set operations.

6.4.1.4 Token Handling

Dual-Token JWT Strategy: Veria implements a sophisticated JWT token system (`packages/auth-middleware/src/index.ts` , `services/identity-service/src/auth/jwt.ts`) balancing security and user experience:

Token Type	TTL	Purpose	Secret
Access Token	15 minutes	API authorization	JWT_SECRET
Refresh Token	7 days	Token renewal	JWT_REFRESH_SECRET

Token Payload Structure: Access tokens include comprehensive authorization data:

- User identification (userId, email)
- Role-based access (role, permissions array)

- Session correlation (sessionId)
- Signing algorithm: HMAC SHA-256 (HS256)

Token Blacklisting: The system implements Redis-backed token blacklisting with 24-hour TTL for revoked tokens, using key pattern `blacklist:<token>` to prevent token replay attacks during security incidents.

6.4.1.5 Password Policies

Comprehensive Password Requirements: The platform enforces enterprise-grade password policies (`services/identity-service/src/auth/password.ts`) meeting financial industry standards:

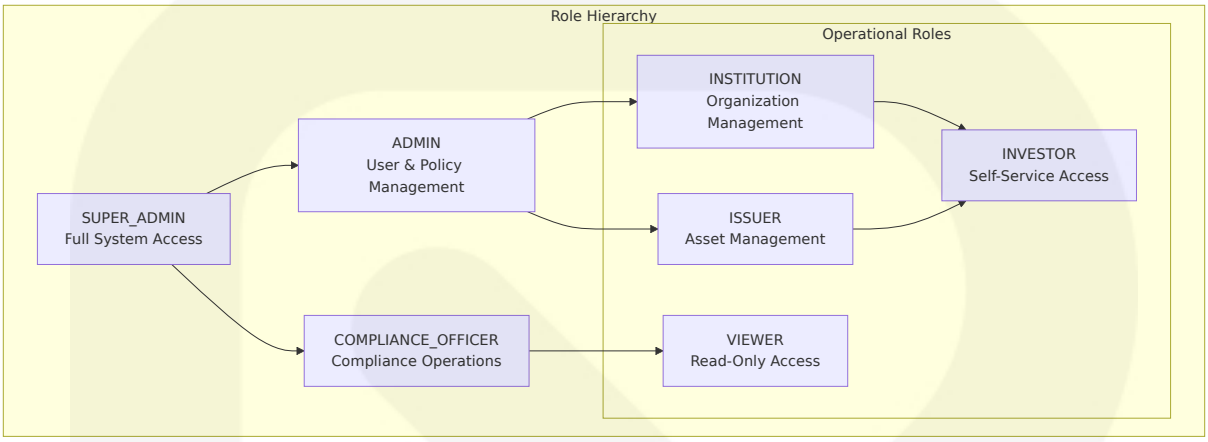
Policy Component	Requirement	Validation
Minimum Length	8 characters	Character count validation
Character Classes	Lowercase, uppercase, digit, special (@$!%*?&)	Regex pattern matching
Forbidden Patterns	No spaces, common passwords	Blacklist checking
Hashing Algorithm	Bcrypt with 10 salt rounds	Cryptographically secure

Password Validation Flow: The system performs real-time password strength validation during registration and password changes, providing immediate feedback to users while maintaining security requirements for compliance environments.

6.4.2 AUTHORIZATION SYSTEM

6.4.2.1 Role-Based Access Control (RBAC)

Hierarchical Role Structure: Veria implements a comprehensive RBAC system (`services/identity-service/src/auth/rbac.ts`) with seven distinct user roles tailored to compliance workflows:



Role Capabilities Matrix: Each role provides specific operational capabilities within the compliance framework:

Role	User Management	Compliance Operations	Asset Management	System Administration
SUPER_ADMIN	Full CRUD	Override capabilities	Full control	System configuration
ADMIN	User CRUD	Policy management	Read/Update	User administration
COMPLIANCE_OFFICER	Review operations	KYC/AML processing	Compliance review	Audit access
INSTITUTION	Organization users	Organization compliance	Organization assets	Organization settings

6.4.2.2 Permission Management

Granular Permission System: The platform implements 50+ granular permissions enabling fine-grained access control across all system operations:

User Management Permissions:

- USER_CREATE, USER_READ, USER_UPDATE, USER_DELETE
- User profile and credential management

Compliance and KYC Permissions:

- COMPLIANCE_READ, COMPLIANCE_WRITE, COMPLIANCE_OVERRIDE
- KYC_REVIEW, KYC_APPROVE, KYC_REJECT, KYC_UPDATE
- COMPLIANCE_REVIEW, COMPLIANCE_APPROVE, COMPLIANCE_REJECT

Transaction and Asset Permissions:

- TRANSACTION_CREATE, TRANSACTION_READ, TRANSACTION_APPROVE, TRANSACTION_CANCEL
- POLICY_CREATE, POLICY_READ, POLICY_UPDATE, POLICY_DELETE, POLICY_EVALUATE

System Administration Permissions:

- SYSTEM_CONFIG, SYSTEM_MONITOR, SYSTEM_BACKUP
- AUDIT_READ, AUDIT_EXPORT

6.4.2.3 Resource Authorization

API-Level Authorization: The auth middleware (`packages/auth-middleware/src/index.ts`) implements factory-pattern permission checking at the route level, ensuring every API endpoint validates appropriate permissions before processing requests.

Service-to-Service Authentication: Internal service communication uses API key authentication with x-api-key header validation:

- Key format: `veria_<timestamp>_<random>`
- Database-backed key validation with associated permissions
- Service-specific key scoping for minimal privilege access

6.4.2.4 Policy Enforcement Points

Gateway-Level Enforcement: The Gateway service acts as the primary policy enforcement point, validating authentication and authorization for all external requests before routing to internal services.

Service-Level Validation: Each internal service implements secondary authorization validation, creating defense-in-depth security architecture with multiple enforcement layers.

Database-Level Security: Role-based database access ensures data-level security even if application-layer authorization is bypassed, with read-only connections for reporting services.

6.4.2.5 Rate Limiting and Audit Logging

Multi-Tier Rate Limiting: The platform implements comprehensive rate limiting using Redis-backed sliding window counters:

Enforcement Level	Default Limit	Window	Key Pattern
User-Based	100 requests	60 seconds	`rate:<userId>`
IP-Based	1000 requests	60 seconds	`rate:<ip>`
Gateway-Level	10,000 requests	60 seconds	Global rate limiting

Authorization Audit Trail: All authentication and authorization events are logged through the audit system with comprehensive context including user identity, resource accessed, action attempted, and decision made.

6.4.3 DATA PROTECTION

6.4.3.1 Encryption Standards

Data at Rest Encryption: The platform implements AES-256 encryption for all sensitive data stored in PostgreSQL databases, ensuring comprehensive protection of investor information, compliance records, and financial data.

Data in Transit Protection: All communications use TLS 1.3 encryption, providing perfect forward secrecy and protection against network-level attacks. This includes client-to-gateway, inter-service, and database connections.

Cryptographic Algorithm Standards: The system uses industry-standard cryptographic algorithms:

- **Symmetric Encryption**: AES-256-GCM for data at rest
- **Asymmetric Signatures**: ES256 and RS256 for WebAuthn
- **Hash Functions**: SHA-256 for audit integrity, Bcrypt for passwords

6.4.3.2 Key Management

Google Secret Manager Integration: Production deployments leverage Google Cloud Secret Manager (`cloudrun.yaml`) for secure key storage and rotation:

Secret Type	Secret Name	Rotation Policy
Database Credentials	db-url	90 days
JWT Signing Keys	jwt-secret, jwt-refresh-secret	30 days
External API Keys	alchemy-api-key, stripe-secret-key	Provider-dependent
Monitoring Keys	sentry-dsn-backend, datadog-api-key	Annual

Development Environment: Local development uses `.env` files (excluded from source control) with example configuration in `.env.example`

for secure local development without compromising production secrets.

Service Account Security: Production services use the dedicated service account `veria-api@${GCP_PROJECT_ID}.iam.gserviceaccount.com` with minimal required permissions for secret access.

6.4.3.3 Data Masking and PII Protection

Personally Identifiable Information (PII) Protection: The platform implements comprehensive PII protection throughout the system:

- **KYC Data Masking**: Sensitive customer information is masked in logs and non-production environments
- **Audit Log PII Redaction**: Personal information is automatically redacted from audit trails while maintaining compliance traceability
- **Database Field Encryption**: Sensitive fields are encrypted at the column level using application-layer encryption

Data Classification Framework: The system classifies data into protection levels:

- **Public**: Non-sensitive configuration and system information
- **Internal**: Business logic and non-personal operational data
- **Confidential**: Customer PII, financial transactions, compliance decisions
- **Restricted**: Cryptographic keys, administrative credentials

6.4.3.4 Secure Communication

Inter-Service Communication Security: All internal service communication occurs over encrypted channels with mutual TLS authentication in production environments.

API Security Headers: The Gateway service implements comprehensive security headers through Fastify Helmet integration:

- Content Security Policy (CSP)

- HTTP Strict Transport Security (HSTS)
- X-Frame-Options for clickjacking protection
- X-Content-Type-Options to prevent MIME sniffing

Request Validation: All API endpoints implement input validation using Zod schemas, preventing injection attacks and ensuring data integrity throughout the system.

6.4.3.5 Compliance Controls

SOX Compliance Support: The audit trail system provides tamper-evident logging capabilities supporting Sarbanes-Oxley compliance requirements for financial reporting.

GDPR Data Protection: The platform supports European data protection requirements through:

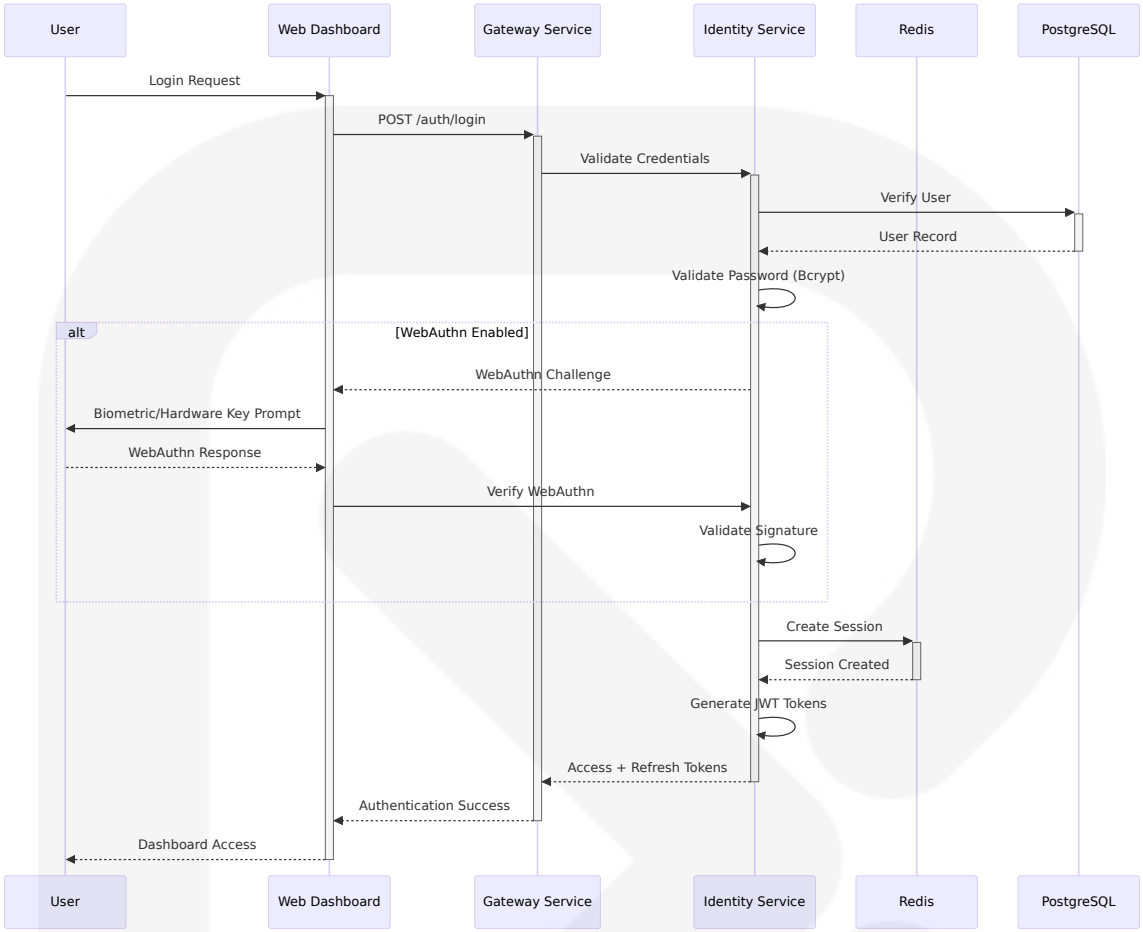
- Data subject access rights via audit trail queries
- Right to erasure implementation in user management
- Data processing transparency through comprehensive logging

Financial Regulatory Compliance: The system supports various financial regulations through:

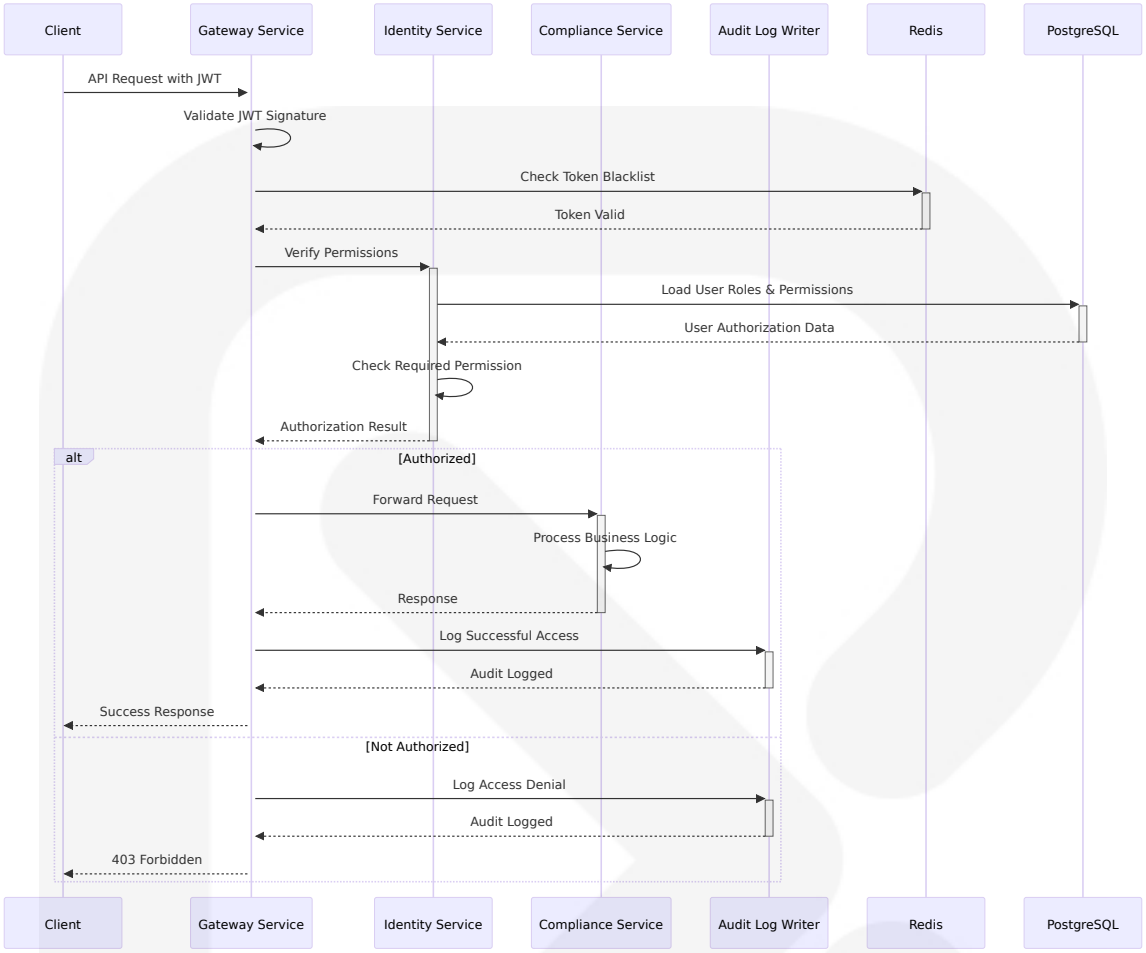
- Comprehensive audit trails for regulatory examinations
- Data retention policies meeting regulatory requirements
- Export capabilities for compliance reporting

6.4.4 SECURITY ARCHITECTURE DIAGRAMS

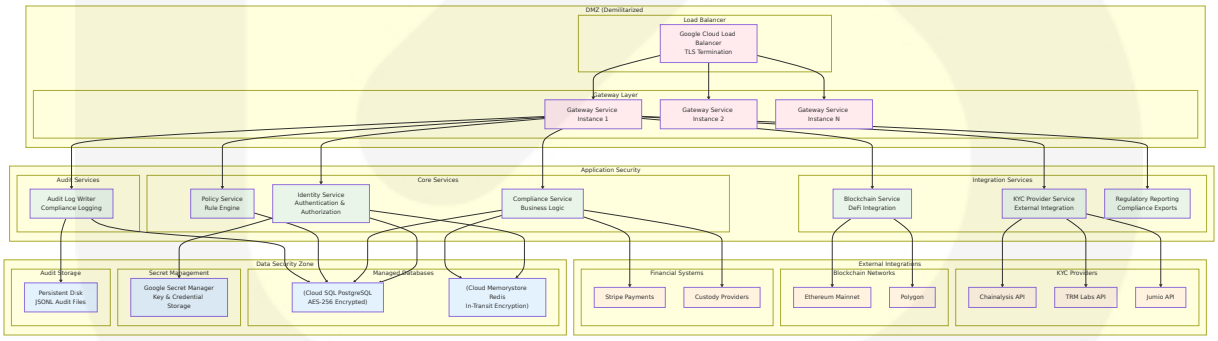
6.4.4.1 Authentication Flow Diagram



6.4.4.2 Authorization Flow Diagram



6.4.4.3 Security Zone Architecture



6.4.5 SECURITY CONTROL MATRICES

6.4.5.1 Access Control Matrix

User Role	User Mgmt	Compliance Ops	KYC/AML	Transaction Ops
SUPER_ADMIN	Full CRUD	Override + Approve	Full Review + Approve	Full Control + Cancel
ADMIN	Create + Update	Policy Management	Review Access	Read + Approve
COMPLIANCE_OFFICER	Read + Update	Review + Approve/Reject	Full KYC Operations	Read + Block
ISSUER	Organization Only	Policy Create + Update	Organization KYC	Create + Update

6.4.5.2 Data Protection Control Matrix

Data Classification	Encryption	Access Control	Audit Level	Retention
Public	TLS in Transit	None Required	Basic Logging	1 Year
Internal	TLS in Transit	Authentication Required	Standard Audit	3 Years
Confidential	AES-256 at Rest + TLS	Authorization Required	Enhanced Audit	7 Years
Restricted	AES-256 + HSM	Admin Only	Full Audit Trail	7 Years + Archive

6.4.5.3 Network Security Controls

Security Layer	Implementation	Monitoring	Response
Perimeter	Cloud Load Balancer + WAF	DDoS Detection	Auto-scaling + Blacklisting
Application	Gateway Service Rate Limiting	Request Pattern Analysis	Circuit Breaker + Alerts
Service Mesh	mTLS + Service Discovery	Inter-service Monitoring	Health Check + Failover

Security Layer	Implementation	Monitoring	Response
Database	Connection Encryption + Pooling	Query Performance + Access	Connection Throttling

6.4.6 COMPLIANCE AND REGULATORY ALIGNMENT

6.4.6.1 Financial Services Compliance

SOX Compliance Framework: The audit trail system supports Sarbanes-Oxley compliance through:

- Immutable audit logging with hash verification
- Executive certification support through comprehensive trail
- Financial reporting integrity through transaction audit
- 7-year retention policy meeting regulatory requirements

Anti-Money Laundering (AML) Support: KYC integration provides AML compliance through:

- Customer identification program (CIP) implementation
- Suspicious activity monitoring and reporting
- Enhanced due diligence for high-risk customers
- Integration with sanctions screening databases

6.4.6.2 Data Privacy Compliance

GDPR Compliance Implementation: European data protection support includes:

- Data subject access rights through audit queries
- Right to erasure via secure deletion processes
- Data processing transparency through audit trails
- Cross-border data transfer controls

Privacy by Design Principles: The platform implements privacy by design through:

- Data minimization in collection and processing
- Purpose limitation through role-based access
- Storage limitation via automated retention policies
- Security by design through comprehensive encryption

6.4.6.3 Industry-Specific Requirements

Securities Regulations: Support for SEC and similar regulatory requirements:

- Investor accreditation verification and audit
- Transaction reporting and compliance monitoring
- Custody provider integration and oversight
- Regulatory filing support and documentation

Tokenization Compliance: Specialized controls for digital asset compliance:

- Smart contract audit trail integration
- On-chain/off-chain data synchronization
- Token transfer compliance verification
- Regulatory reporting for tokenized securities

References

Repository Files Examined

- `packages/auth-middleware/src/index.ts` - Authentication middleware implementation with JWT validation and rate limiting
- `services/identity-service/src/auth/webauthn.ts` - WebAuthn/FIDO2 multi-factor authentication implementation
- `services/identity-service/src/auth/session.ts` - Redis-backed distributed session management system

- `services/identity-service/src/auth/rbac.ts` - Role-based access control with comprehensive permission matrix
- `services/identity-service/src/auth/password.ts` - Password policy enforcement and bcrypt hashing
- `services/identity-service/src/auth/jwt.ts` - Dual-token JWT implementation with blacklisting support
- `services/audit-log-writer/src/index.ts` - Dual-write audit trail implementation for compliance logging
- `.env.example` - Development environment security configuration template
- `cloudrun.yaml` - Production security configuration with Google Secret Manager integration

Repository Folders Analyzed

- `` - Repository root structure and security configuration overview
- `packages/auth-middleware/` - Shared authentication and authorization middleware for all services
- `services/identity-service/src/auth/` - Complete authentication framework implementation
- `services/audit-log-writer/` - Compliance audit trail system with dual-write pattern
- `services/gateway/` - Gateway service security controls and rate limiting implementation

Technical Specification Sections Referenced

- `2.5 NON-FUNCTIONAL REQUIREMENTS` - Security requirements including encryption standards and authentication
- `5.4 CROSS-CUTTING CONCERNS` - Authentication framework, audit patterns, and monitoring implementation
- `6.1 CORE SERVICES ARCHITECTURE` - Service mesh security, health checks, and resilience patterns

6.5 MONITORING AND OBSERVABILITY

6.5.1 MONITORING INFRASTRUCTURE

6.5.1.1 Current Implementation Status

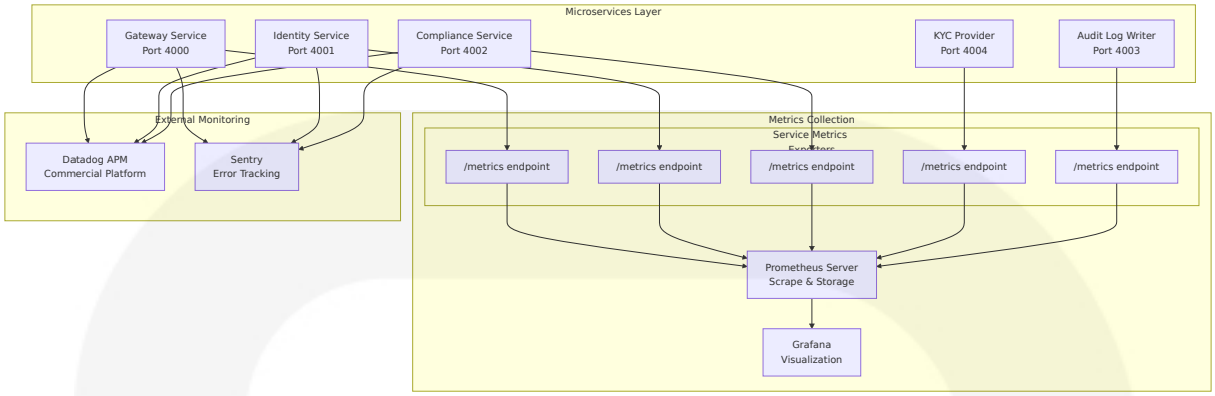
Veria implements a **hybrid monitoring architecture** combining partial observability features with planned enterprise-grade monitoring infrastructure. The current state reflects a system in active development with foundational monitoring capabilities established and advanced features planned for production readiness.

Implementation Matrix:

Component	Status	Implementation Details	Production Readiness
Health Checks	âœ… Implemented	All services expose `/health` endpoints	Production-ready
Request Tracing	âœ… Implemented	x-request-id propagation across services	Production-ready
Structured Logging	âš ï¸ Partial	Pino logger configured, not fully deployed	Development-ready
Metrics Collection	â Planned	Prometheus-compatible metrics documented	Implementation required

6.5.1.2 Metrics Collection Architecture

The monitoring system follows the three pillars of observability: logs, metrics, and traces, providing comprehensive visibility into the distributed system. The planned metrics collection infrastructure implements a **push-based Prometheus architecture** with service discovery and multi-tier aggregation.



Planned Metrics Categories

Application Performance Metrics:

Metric Type	Description	Collection Method	Retention Period
HTTP Request Duration	Response time percentiles (p50, p95, p99)	Prometheus histograms	30 days detailed, 1 year aggregated
Request Rate	Requests per second by endpoint and status	Prometheus counters	30 days detailed, 1 year aggregated
Error Rate	4xx/5xx response percentage	Prometheus counters	90 days detailed, 2 years aggregated
Database Connection Pool	Active/idle connections, wait time	Custom gauges	7 days detailed, 90 days aggregated

Business Metrics:

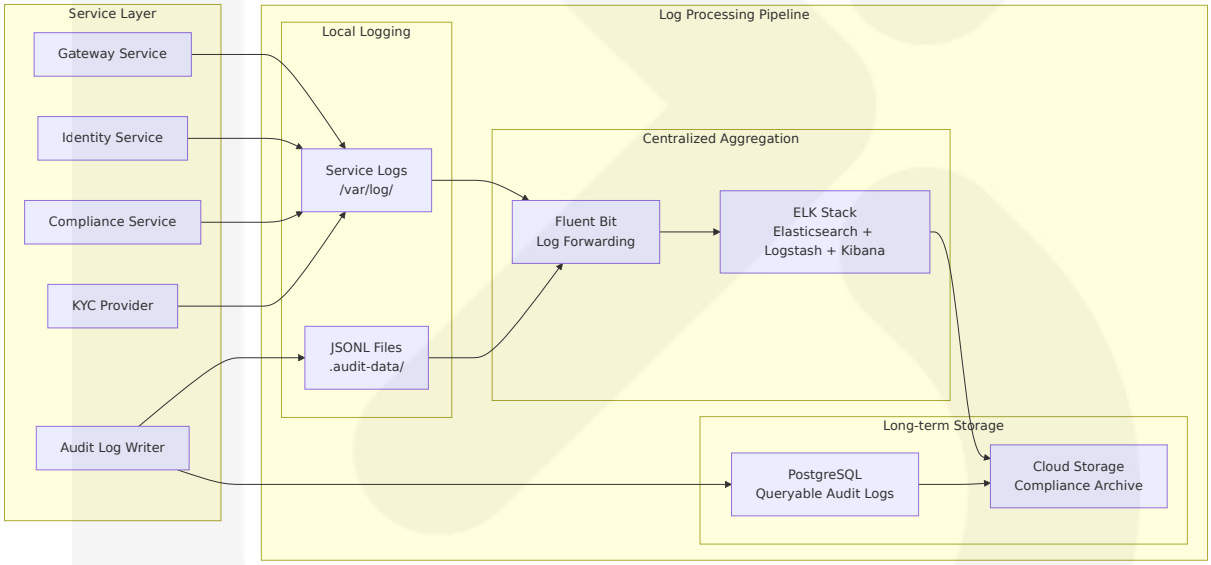
Metric Type	Description	Business Impact	Alert Threshold
Compliance Decision Rate	Decisions per minute by outcome	Revenue-affecting	<100 decisions/min
KYC Processing Time	End-to-end verification duration	Customer experience	>30s p95

Metric Type	Description	Business Impact	Alert Threshold
Audit Log Write Success	Dual-write pattern success rate	Regulatory compliance	<99.9% success
Rate Limiting Violations	IP-based throttling events	System protection	>1000 violations/hour

6.5.1.3 Log Aggregation Strategy

The logging architecture implements structured logging as the foundation of microservices observability, utilizing **dual-write audit patterns** and **centralized log aggregation** for both operational and compliance requirements.

Logging Infrastructure Design:



Log Level Configuration:

Service	Debug Level	Info Level	Warn Level	Error Level
Gateway Service	Request routing details	Authentication events	Rate limit warnings	Proxy failures
Identity Service	JWT token validation	User login/logout	Session expiration	Authentication failures

Service	Debug Level	Info Level	Warn Level	Error Level
Compliance Service	Rule evaluation steps	Decision outcomes	Threshold violations	Rule engine errors
KYC Provider	Provider selection logic	Verification results	Fallback activations	Provider failures

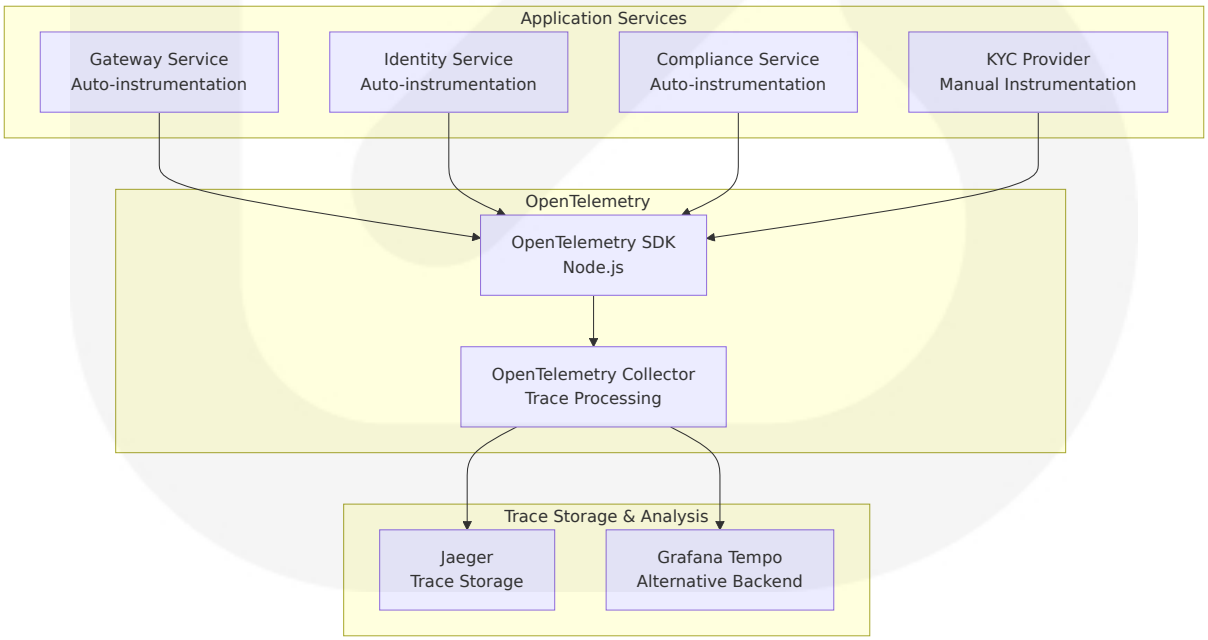
6.5.1.4 Distributed Tracing Implementation

Veria implements **request ID-based distributed tracing** with planned migration to **OpenTelemetry standards** for enhanced trace correlation and analysis.

Current Tracing Architecture:

The system generates correlation IDs using the pattern `req_<timestamp><8-char-random>` through the Gateway service's `reqid.js` implementation. Request IDs flow through the entire service mesh, enabling end-to-end request tracking across all microservice boundaries.

Planned OpenTelemetry Integration:



6.5.1.5 Alert Management Infrastructure

The monitoring system provides intelligent alerting mechanisms that prioritize critical issues and reduce noise, implementing a **multi-tier alerting architecture** with escalation procedures and incident response automation.

Alert Routing Configuration:

Alert Category	Initial Response	Escalation Time	Escalation Target	Communication Channel
System Critical	Immediate page	5 minutes	Senior DevOps	PagerDuty + SMS
Compliance Violation	Immediate notification	15 minutes	Compliance Team	Email + Slack
Performance Degradation	Slack notification	30 minutes	Development Team	Slack + Email
Security Incident	Immediate page	2 minutes	Security Team	PagerDuty + SMS + Email

6.5.2 OBSERVABILITY PATTERNS

6.5.2.1 Health Check Architecture

All services implement **comprehensive health monitoring** through standardized `/health` endpoints with multi-dimensional health assessments covering database connectivity, cache availability, and external service dependencies.

Health Check Implementation Matrix:

Service	Endpoint	Dependencies Checked	Response Format	Cloud Run Integration
Gateway Service	GET /health	None (stateless proxy)	`{status: 'ok', name: 'gateway', ts: ISO}`	âœ… Configured
Identity Service	GET /health	PostgreSQL, Redis	Service-specific health object	âœ… Configured
Compliance Service	GET /health	PostgreSQL, Redis, Policy Service	Service-specific health object	âœ… Configured
KYC Provider	GET /health	PostgreSQL, Redis, External APIs	Multi-provider status object	âœ… Configured

Cloud Run Health Check Configuration:

```
livenessProbe:
  httpGet:
    path: /health
    port: 8080
  initialDelaySeconds: 30
  periodSeconds: 30
  failureThreshold: 3

readinessProbe:
  httpGet:
    path: /health
    port: 8080
  initialDelaySeconds: 10
  periodSeconds: 10
  failureThreshold: 3
```

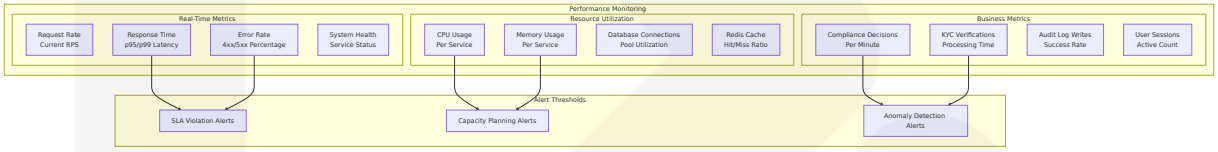
6.5.2.2 Performance Metrics Monitoring

Service Level Objectives (SLOs) Implementation:

Based on the technical specification requirements, Veria implements **strict SLA monitoring** with automated alerting for threshold violations and capacity planning triggers.

Component	Availability SLA	Response Time SLA	Throughput SLA	Error Rate SLA
Gateway Service	99.99%	<50ms (p95)	10,000+ req/sec	<0.1%
Identity Service	99.95%	<100ms (p95)	5,000+ auth/sec	<0.5%
Compliance Service	99.90%	<200ms (p95)	1,000+ decisions/sec	<1.0%
KYC Provider Service	99.50%	<5s (p95)	100+ verifications/sec	<2.0%

Performance Monitoring Dashboard Design:



6.5.2.3 Business Metrics Tracking

Compliance Operations Monitoring:

Metric	Measurement	Target Value	Alert Threshold	Business Impact
Compliance Decision Latency	Time from request to decision	<100ms p95	>200ms p95	Customer experience degradation
Rule Engine Evaluation Rate	Rules processed per second	>1000 evaluations/sec	<500 evaluations/sec	Processing bottleneck
Audit Trail Completeness	Percentage of events logged	100%	<99.9%	Regulatory compliance risk

Metric	Measurement	Target Value	Alert Threshold	Business Impact
Cross-Service Request Success	Inter-service call success rate	>99.5%	<99.0%	Service mesh degradation

6.5.2.4 Capacity Tracking and Auto-Scaling

Auto-Scaling Monitoring Architecture:

Cloud Run auto-scaling configuration with **intelligent capacity monitoring** based on multiple metrics including CPU utilization, memory pressure, request queue depth, and response latency.

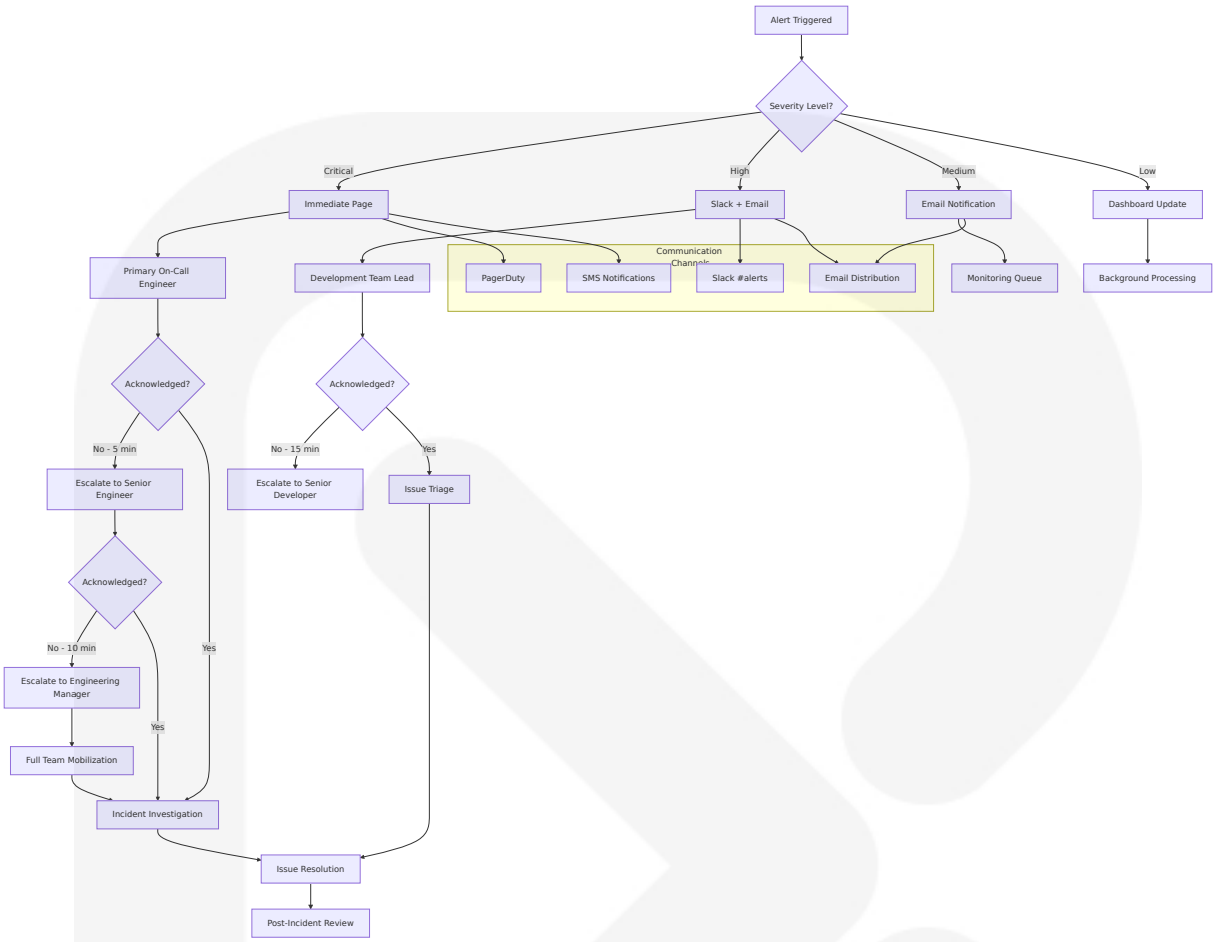
Scaling Trigger Matrix:

Metric	Scale-Up Threshold	Scale-Down Threshold	Action	Cooldown Period
CPU Utilization	>70% average	<30% average	Add/remove 1 instance	10 minutes
Memory Usage	>80% average	<40% average	Add/remove 1 instance	5 minutes
Request Queue Depth	>10 requests	<2 requests	Add/remove 2 instances	2 minutes
Response Latency	>500ms p95	<200ms p95	Add/remove 1 instance	15 minutes

6.5.3 INCIDENT RESPONSE

6.5.3.1 Alert Routing and Escalation

Multi-Tier Incident Response Framework:



6.5.3.2 Escalation Procedures

Incident Severity Classification:

Severity	Definition	Response Time	Escalation Path	Communication
Critical (P0)	Service completely unavailable	Immediate (0-5 min)	On-call â†’ Senior â†’ Manager	PagerDuty + SMS + Slack
High (P1)	Significant performance degradation	15 minutes	Team Lead â†’ Senior Developer	Slack + Email
Medium (P2)	Minor performance issues	1 hour	Development queue	Email

Severity	Definition	Response Time	Escalation Path	Communication
Low (P3)	Monitoring anomalies	4 hours	Background triage	Dashboard

Incident Command Structure:

1. **Incident Commander**: Senior on-call engineer responsible for coordination
2. **Technical Lead**: Service owner with deep system knowledge
3. **Communications Lead**: Stakeholder communication and status updates
4. **Subject Matter Expert**: Domain-specific expertise (compliance, security, infrastructure)

6.5.3.3 Runbook Documentation

Automated Runbook Execution:

Critical incidents trigger automated runbook execution through infrastructure-as-code patterns, reducing mean time to resolution (MTTR) and human error risk.

Service-Specific Runbooks:

Service	Common Issues	Automated Response	Manual Intervention	Recovery Time
Gateway Service	Rate limiting overload	Scale up instances, adjust limits	Review traffic patterns	2-5 minutes
Identity Service	JWT validation failures	Restart service, clear Redis cache	Investigate auth provider	5-10 minutes
Compliance Service	Rule engine timeouts	Increase memory allocation	Review rule complexity	10-15 minutes

Service	Common Issues	Automated Response	Manual Intervention	Recovery Time
KYC Provider	External API failures	Switch to fallback provider	Contact vendor support	15-30 minutes

6.5.3.4 Post-Mortem Process

Structured Post-Incident Analysis:

Following AI and machine learning for proactive issue detection and resolution best practices, Veria implements **blameless post-mortem culture** focused on system improvement and prevention.

Post-Mortem Template Structure:

1. **Incident Timeline**: Detailed chronology from detection to resolution
2. **Root Cause Analysis**: Technical and process factors contributing to incident
3. **Impact Assessment**: Quantified business and customer impact
4. **Response Evaluation**: Effectiveness of detection, escalation, and resolution
5. **Improvement Actions**: Specific, measurable prevention and response improvements

Improvement Tracking Matrix:

Action Item	Owner	Target Date	Success Metric	Status
Implement circuit breakers for KYC service	Backend Team	30 days	Reduced cascade failures	In Progress
Add business metrics alerting	DevOps Team	14 days	Faster incident detection	Not Started
Automate database failover	Infrastructure Team	45 days	<30s recovery time	Planning

Action Item	Owner	Target Date	Success Metric	Status
Enhance monitoring dashboards	Product Team	21 days	Improved observability	In Progress

6.5.3.5 Continuous Improvement Framework

Monitoring Evolution Strategy:

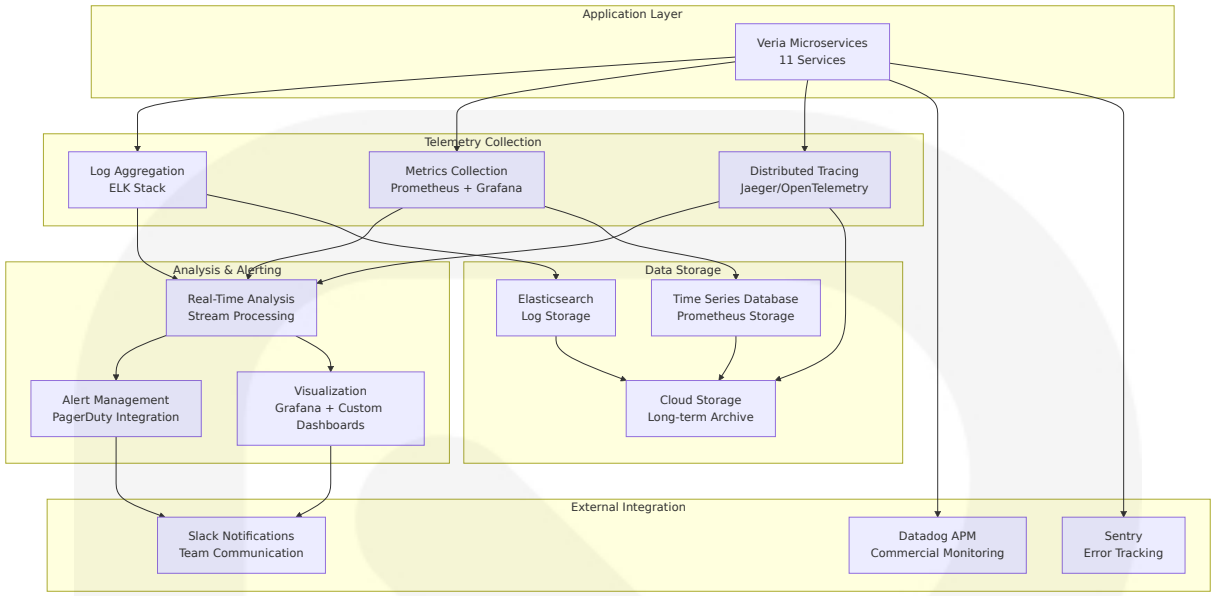
The observability strategy incorporates continuous improvement practices as systems evolve, implementing regular monitoring reviews and capability assessments.

Quarterly Review Process:

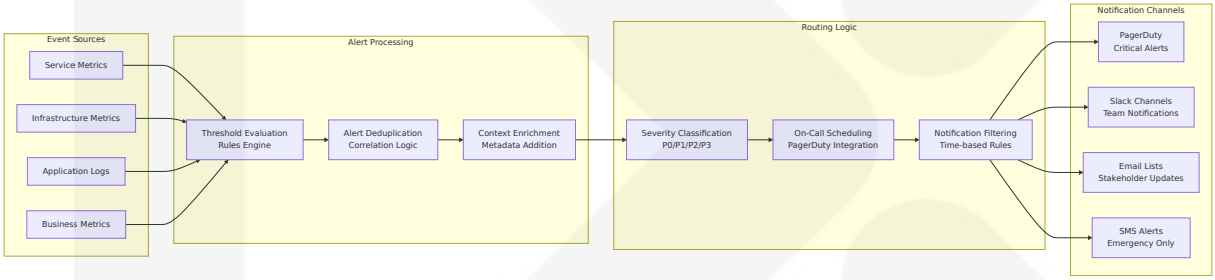
1. **Alert Effectiveness Analysis**: Review alert noise ratio and false positive rates
2. **SLA Performance Assessment**: Evaluate actual performance against defined objectives
3. **Incident Pattern Analysis**: Identify recurring issues and systemic problems
4. **Tool Evaluation**: Assess monitoring tool effectiveness and potential improvements
5. **Team Training Updates**: Update incident response procedures and knowledge sharing

6.5.4 MONITORING ARCHITECTURE DIAGRAMS

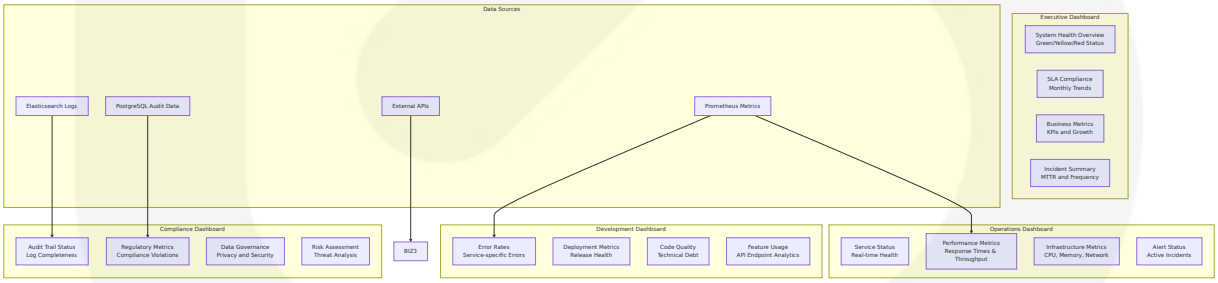
6.5.4.1 Complete Monitoring Ecosystem



6.5.4.2 Alert Flow Architecture



6.5.4.3 Dashboard Layout Architecture



References

Repository Files Examined

- `services/gateway/src/reqid.js` - Request ID generation and propagation implementation
- `services/gateway/src/server.js` - Gateway health check and proxy configuration
- `tests/performance/load-test.js` - k6 performance testing with SLA thresholds
- `services/audit-log-writer/` - Dual-write audit pattern implementation
- `cloudrun.yaml` - Production deployment with Sentry/Datadog integration
- `docker-compose.yml` - Container health check configuration
- `.env.example` - Environment configuration template with logging settings

Repository Folders Explored

- `` (root) - Repository structure and monorepo configuration
- `services/` (11 services) - Microservices architecture overview
- `services/gateway/` - Gateway service implementation and health checks
- `services/compliance-service/` - Compliance service monitoring patterns
- `services/identity-service/` - Identity service health and performance monitoring
- `packages/auth-middleware/` - Shared authentication monitoring utilities
- `.github/workflows/` - CI/CD monitoring and deployment health checks
- `ops/` - Operational runbooks and monitoring procedures
- `infra/` - Infrastructure monitoring and deployment configurations

Technical Specification Sections Referenced

- `2.5 NON-FUNCTIONAL REQUIREMENTS` - Performance requirements and SLA definitions
- `5.4 CROSS-CUTTING CONCERNS` - Detailed monitoring and observability patterns

- `6.1 CORE SERVICES ARCHITECTURE` - Service health and resilience patterns
- `4.3 PERFORMANCE AND MONITORING FLOWS` - Real-time monitoring pipeline design

Web Search Sources

- Last9.io observability best practices for microservices standardization
- SigNoz guide on essential microservices monitoring in 2025
- Lumigo microservices observability pillars and patterns
- OpenObserve mastering microservices monitoring tools for 2024
- Cerbos monitoring and observability challenges and best practices

6.6 TESTING STRATEGY

6.6.1 TESTING APPROACH OVERVIEW

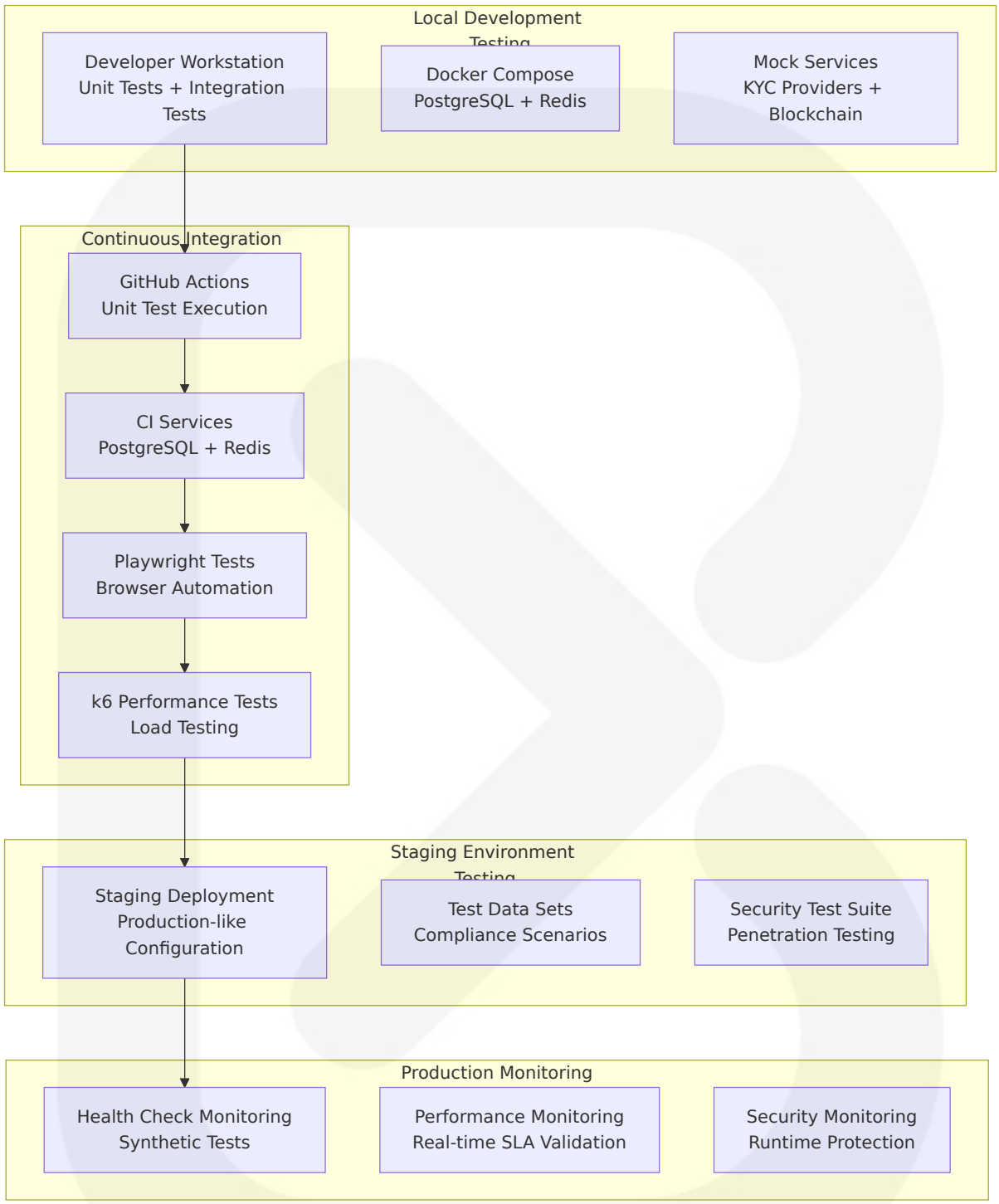
Veria's testing strategy implements a comprehensive multi-layered approach designed to ensure regulatory compliance, performance standards, and security requirements are met across the entire tokenized asset compliance platform. The strategy encompasses unit testing for individual components, integration testing for service interactions, end-to-end testing for complete workflows, and specialized testing for performance, security, and compliance requirements.

The testing approach is built around the **quality pyramid principle** with extensive unit tests forming the foundation, focused integration tests ensuring service interactions work correctly, and targeted end-to-end tests validating critical compliance workflows. This approach provides rapid feedback during development while maintaining confidence in production deployments.

6.6.1.1 Testing Framework Ecosystem

Testing Category	Primary Framework	Secondary Framework	Target Coverage	Execution Environment
Unit Testing	Vitest (Node.js services)	pytest (Python packages)	85%+ line coverage	Local development + CI
Integration Testing	Vitest with axios clients	Hardhat Test (smart contracts)	90%+ service interactions	Docker Compose + CI
End-to-End Testing	Playwright (browser tests)	Vitest (API workflows)	100% critical paths	Dedicated test environments
Performance Testing	k6 with defined thresholds	Vitest (API load tests)	SLA compliance validation	Load testing environment
Security Testing	Custom security test suites	Trivy vulnerability scanning	100% security controls	Secure test environment

6.6.1.2 Test Environment Architecture



6.6.2 UNIT TESTING

6.6.2.1 Testing Frameworks and Tools

Primary Framework: Vitest Configuration

Veria utilizes **Vitest** as the primary unit testing framework for all Node.js services, chosen for its native TypeScript support, ESM compatibility, and excellent performance characteristics. The configuration implements comprehensive coverage reporting with V8 provider for accurate Node.js coverage measurement.

Vitest Configuration Standards:

```
// Example configuration pattern used across services
export default defineConfig({
  test: {
    globals: true,
    environment: 'node',
    coverage: {
      provider: 'v8',
      reporter: ['text', 'json', 'html'],
      exclude: [
        'coverage/**',
        'dist/**',
        'node_modules/**',
        '**/*.d.ts',
        '**/*.config.*',
        '**/tests/**'
      ]
    }
  }
});
```

Secondary Framework: pytest for Python Components

The blockchain package utilizes **pytest** for Python-based smart contract interaction testing and compliance algorithm validation, providing specialized support for Web3 integration testing.

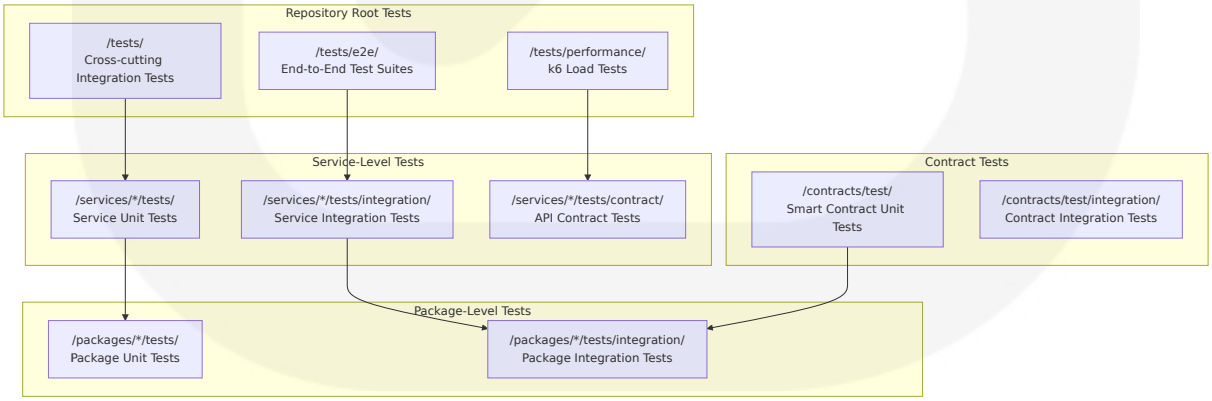
Framework Selection Matrix:

Service Type	Testing Framework	Configuration File	Coverage Target	Mock Strategy
Gateway Service	Vitest	`vitest.config.ts`	90%+	Request/response mocking
Identity Service	Vitest	`vitest.config.ts`	85%+	Database + Redis mocking
Compliance Service	Vitest	`vitest.config.ts`	90%+	Rule engine mocking
KYC Provider	Vitest	`vitest.config.ts`	80%+	External API mocking
Blockchain Package	pytest	`pytest.ini`	85%+	Web3 provider mocking
Smart Contracts	Hardhat Test	`hardhat.config.js`	95%+	Hardhat network mocking

6.6.2.2 Test Organization Structure

Monorepo Test Architecture:

The testing structure follows the monorepo organization with tests distributed across multiple levels to maintain clear separation of concerns and enable efficient test execution:



Test File Naming Conventions:

Test Type	File Pattern	Location	Execution Command
Unit Tests	`*.test.ts`	`src/` alongside source files	`pnpm test`
Integration Tests	`*.integration.test.ts`	`tests/integration/`	`pnpm test:integration`
Contract Tests	`*.contract.test.ts`	`tests/contract/`	`pnpm test:contract`
End-to-End Tests	`*.e2e.test.ts`	`tests/e2e/`	`pnpm test:e2e`

6.6.2.3 Mocking Strategy

Comprehensive Mocking Architecture:

Veria implements a **layered mocking strategy** that isolates units under test while maintaining realistic behavior patterns for external dependencies and internal service interactions.

Database Mocking:

- **PostgreSQL**: Vitest mocks with in-memory database for fast unit tests
- **Redis**: Custom Redis mock implementation with TTL and data structure support
- **Connection Pooling**: Mock connection pool behavior for resource management testing

External Service Mocking:

- **KYC Providers**: Configurable mock responses for Chainalysis, TRM Labs, and Jumio APIs
- **Blockchain Networks**: Hardhat local network for smart contract testing
- **Payment Processors**: Stripe test mode integration with webhook simulation

Inter-Service Mocking:

- **Gateway Routing**: Mock HTTP clients for service-to-service communication
- **Authentication**: JWT token validation mocking for authorization testing
- **Audit Logging**: Mock audit trail writers to verify logging behavior

6.6.2.4 Code Coverage Requirements

Coverage Targets by Service:

Service Category	Line Coverage Target	Branch Coverage Target	Function Coverage Target	Critical Path Coverage
Core Services	85%+	80%+	90%+	100%
Integration Services	80%+	75%+	85%+	95%
Shared Packages	90%+	85%+	95%+	100%
Smart Contracts	95%+	90%+	100%	100%

Coverage Exclusions:

- Configuration files (`*.config.js` , `*.config.ts`)
- Type definition files (`*.d.ts`)
- Generated code and migrations
- Development utilities and test helpers

6.6.2.5 Test Data Management

Test Data Strategy:

Test data management implements **isolated data sets** for each test suite with automatic cleanup and consistent state management across test runs.

Data Management Patterns:

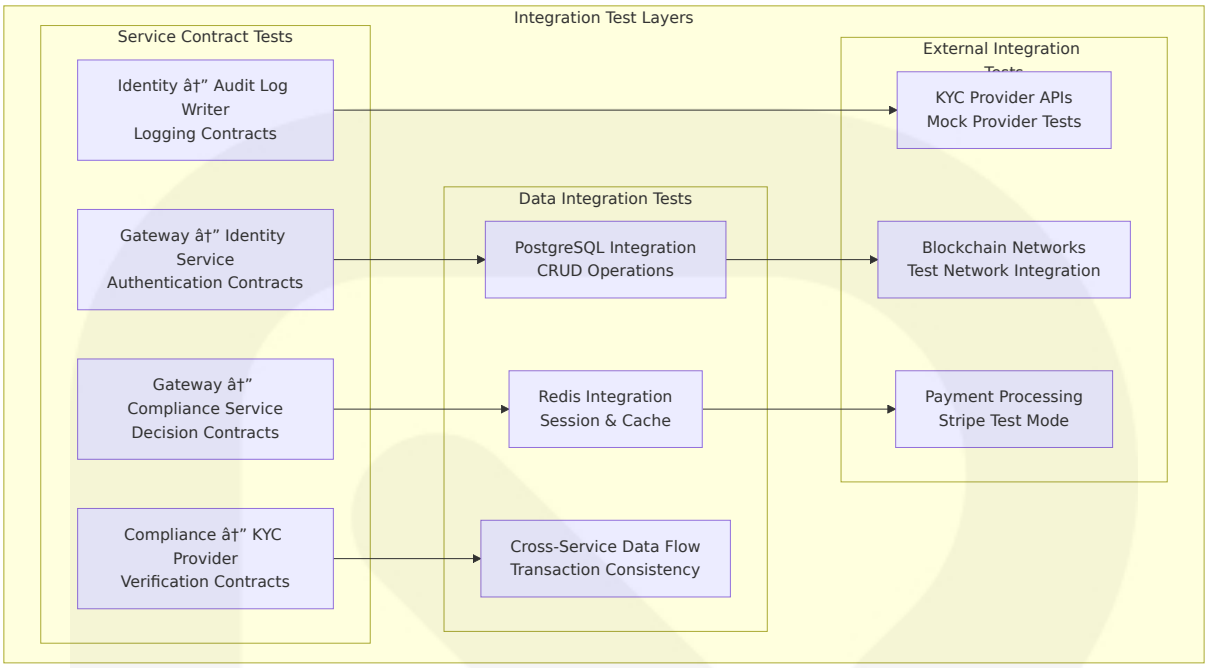
Data Type	Management Strategy	Cleanup Method	Isolation Level
User Test Data	Factory pattern generation	After each test	Per test case
Compliance Scenarios	Predefined scenario sets	After test suite	Per test suite
Transaction Data	Generated sequences	Database rollback	Per test transaction
Audit Trail Data	Mock audit entries	Memory cleanup	Per test run

6.6.3 INTEGRATION TESTING

6.6.3.1 Service Integration Test Approach

Service-to-Service Testing Framework:

Integration testing validates the complete interaction patterns between services, ensuring proper data flow, error handling, and performance characteristics across service boundaries. The approach utilizes **contract testing** principles to verify API compatibility and behavior consistency.



Integration Test Environment Configuration:

Integration tests execute in **Docker Compose environments** with real database instances and Redis cache systems to validate actual service behavior under production-like conditions.

Component	Test Configuration	Purpose	Health Check
PostgreSQL	postgres:15-alpine	Database integration validation	Connection pool testing
Redis	redis:7-alpine	Session and cache integration	TTL and eviction testing
Service Instances	Individual service containers	Cross-service communication	Health endpoint validation

6.6.3.2 API Testing Strategy

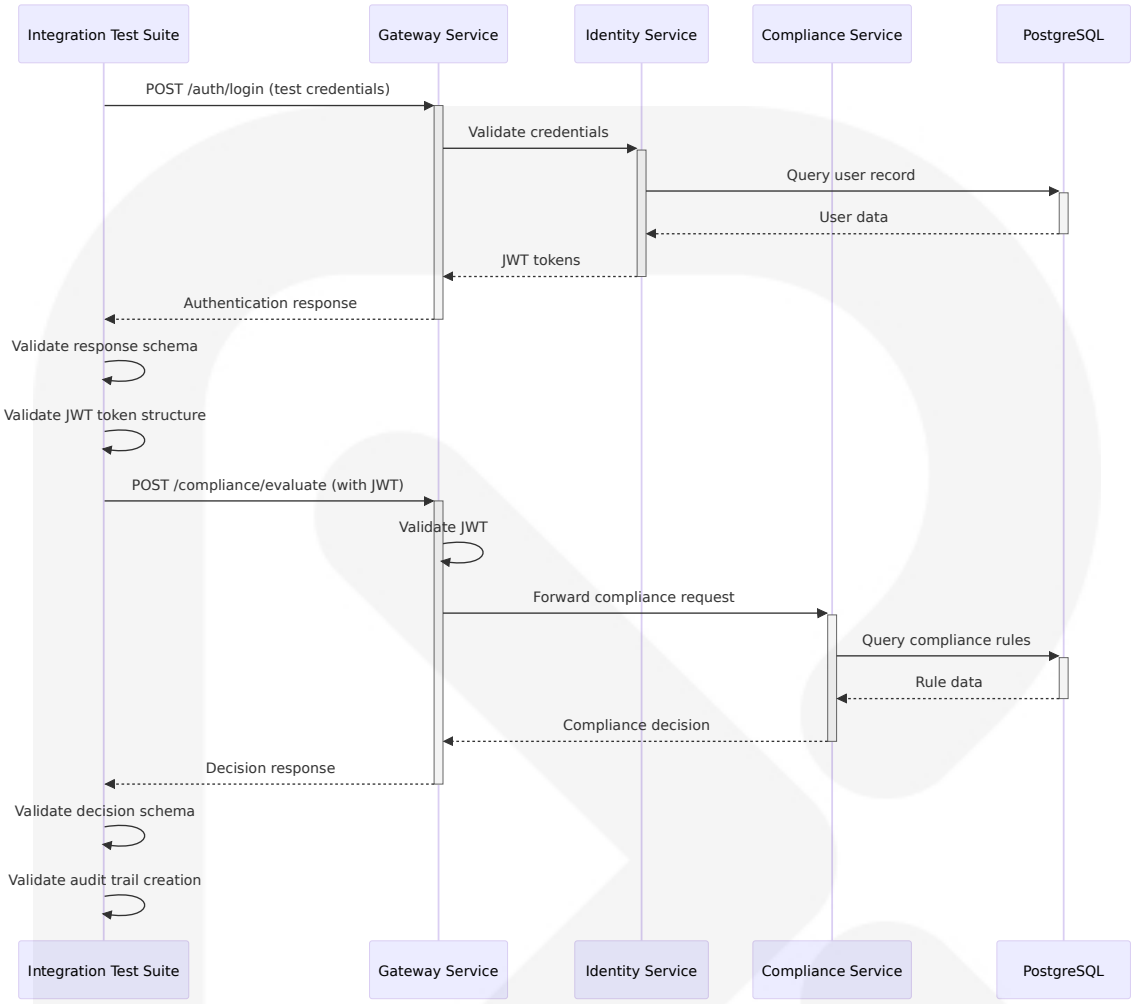
Comprehensive API Test Coverage:

API testing validates all REST endpoints, GraphQL queries, and internal service APIs to ensure proper request handling, response formatting, and error management across the entire platform.

API Test Categories:

Test Category	Scope	Framework	Validation Focus
REST Endpoint Tests	All public APIs	Vitest + axios	Request/response schemas, HTTP status codes
Authentication Tests	JWT validation flows	Vitest + supertest	Token validation, session management
Authorization Tests	RBAC enforcement	Vitest + test users	Permission validation, access control
Error Handling Tests	Error response patterns	Vitest + error scenarios	Error codes, message consistency

API Contract Validation:



6.6.3.3 Database Integration Testing

Database Test Strategy:

Database integration tests validate **transactional consistency**, **constraint enforcement**, and **performance characteristics** across all data access patterns used by the application services.

Database Integration Test Matrix:

Test Type	Coverage	Validation Method	Performance Requirement
CRUD Operations	All entity types	Schema validation + data integrity	<10ms per operation
Transaction Consistency	Cross-table transactions	Rollback testing + isolation	<50ms per transaction
Constraint Enforcement	Foreign key relationships	Error condition testing	Immediate validation
Connection Pool Management	Pool exhaustion scenarios	Resource leak detection	<100 active connections

Database Schema Validation:

Integration tests validate that database schemas support all required operations while maintaining referential integrity and performance requirements established in the non-functional requirements.

6.6.3.4 External Service Mocking

External Integration Mock Strategy:

External service mocking provides **deterministic test environments** while maintaining realistic behavior patterns for third-party integrations including KYC providers, blockchain networks, and payment processors.

Mock Service Implementation:

External Service	Mock Implementation	Test Scenarios	Failure Simulation
Chainalysis KYC	Express server with configurable responses	Address verification, risk scoring	API timeout, rate limiting
TRM Labs	JSON response mocking with axios interceptors	Transaction monitoring, compliance	Network errors, invalid responses

External Service	Mock Implementation	Test Scenarios	Failure Simulation
Jumio Identity	Webhook simulation for document verification	ID verification, biometric matching	Processing failures, expired documents
Ethereum Network	Hardhat local network	Smart contract interactions	Gas estimation errors, revert scenarios

6.6.3.5 Test Environment Management

Environment Lifecycle Management:

Test environments utilize **infrastructure as code** principles with automated provisioning, configuration, and cleanup to ensure consistent and isolated testing conditions.

Environment Configuration Matrix:

Environment	Purpose	Lifespan	Data Management	Resource Allocation
Developer Local	Individual development testing	Persistent	Local Docker volumes	2 CPU, 4GB RAM
CI Integration	Automated integration testing	Per CI run	Ephemeral containers	4 CPU, 8GB RAM
Shared Staging	Manual integration testing	Persistent	Sanitized production subset	8 CPU, 16GB RAM
Performance Testing	Load testing and benchmarking	On-demand	Generated load data	16 CPU, 32 GB RAM

6.6.4 END-TO-END TESTING

6.6.4.1 E2E Test Scenarios

Critical Compliance Workflow Testing:

End-to-end testing validates the three primary business workflows identified in the system requirements: Asset Onboarding, Investor Management, and Compliance Export. These tests ensure complete functionality from user interface through backend processing and data persistence.

Primary E2E Test Scenarios:

Workflow	Test Scenario	Expected Outcome	Performance Target	Compliance Validation
Asset Onboarding	Complete asset registration flow	Asset created with compliance approval	<5 minutes end-to-end	Regulatory document attachment
Investor KYC	Full investor verification process	Investor approved with access credentials	<30 seconds verification	AML/KYC compliance validation
Compliance Export	Generate audit trail export	Signed ZIP with manifest.json	<2 minutes for standard export	Complete audit trail integrity
Transaction Processing	Asset purchase with compliance check	Transaction completed with audit log	<10 seconds processing	RBAC enforcement validation

E2E Test Coverage Matrix:

6.6.4.2 UI Automation Approach

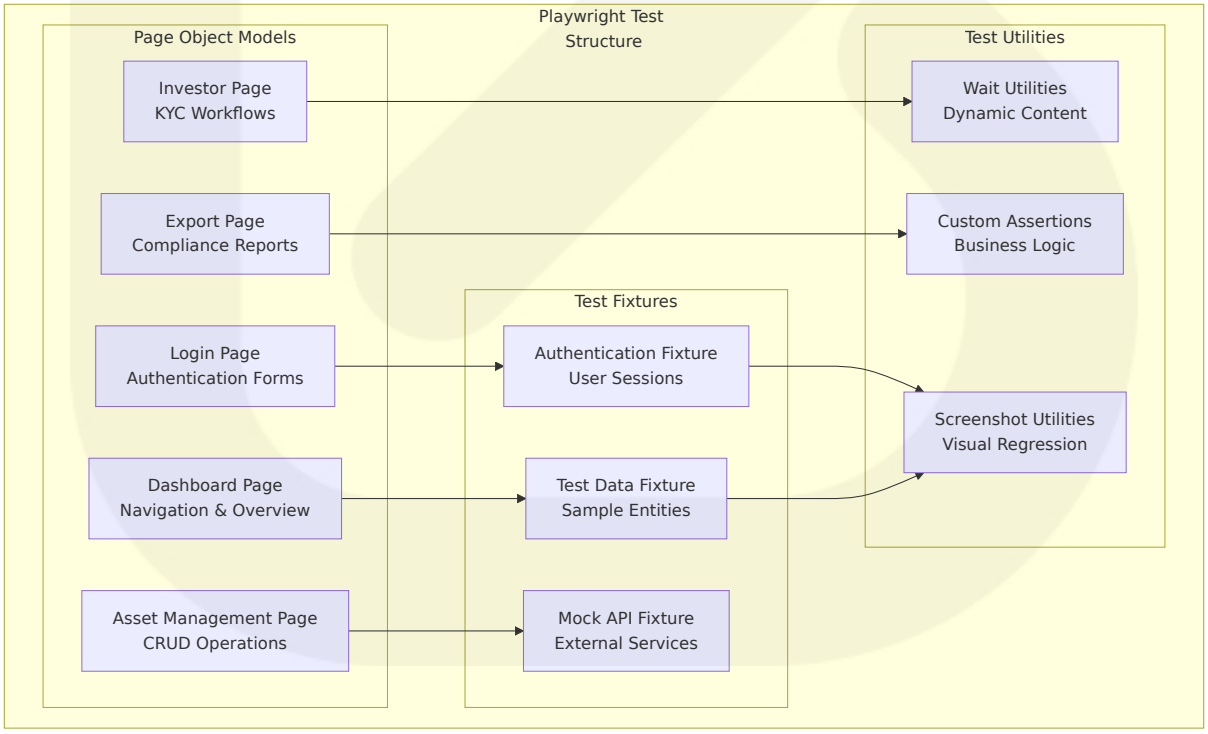
Playwright-Based Browser Testing:

Veria utilizes **Playwright** for comprehensive browser-based UI automation, providing cross-browser compatibility testing and realistic user interaction simulation across the compliance dashboard and main frontend applications.

Playwright Configuration Standards:

Browser Engine	Test Coverage	Viewport Configuration	Performance Threshold
Chromium	Primary test execution	1920x1080 desktop	<3s page load
Firefox	Cross-browser validation	1366x768 standard	<4s page load
WebKit	Safari compatibility	1440x900 macOS	<3.5s page load
Mobile Chrome	Mobile responsiveness	375x667 mobile	<5s page load

UI Test Architecture:



Visual Regression Testing:

Playwright implements **visual regression testing** to detect unintended UI changes that could affect compliance workflow usability or accessibility requirements.

6.6.4.3 Test Data Setup/Teardown

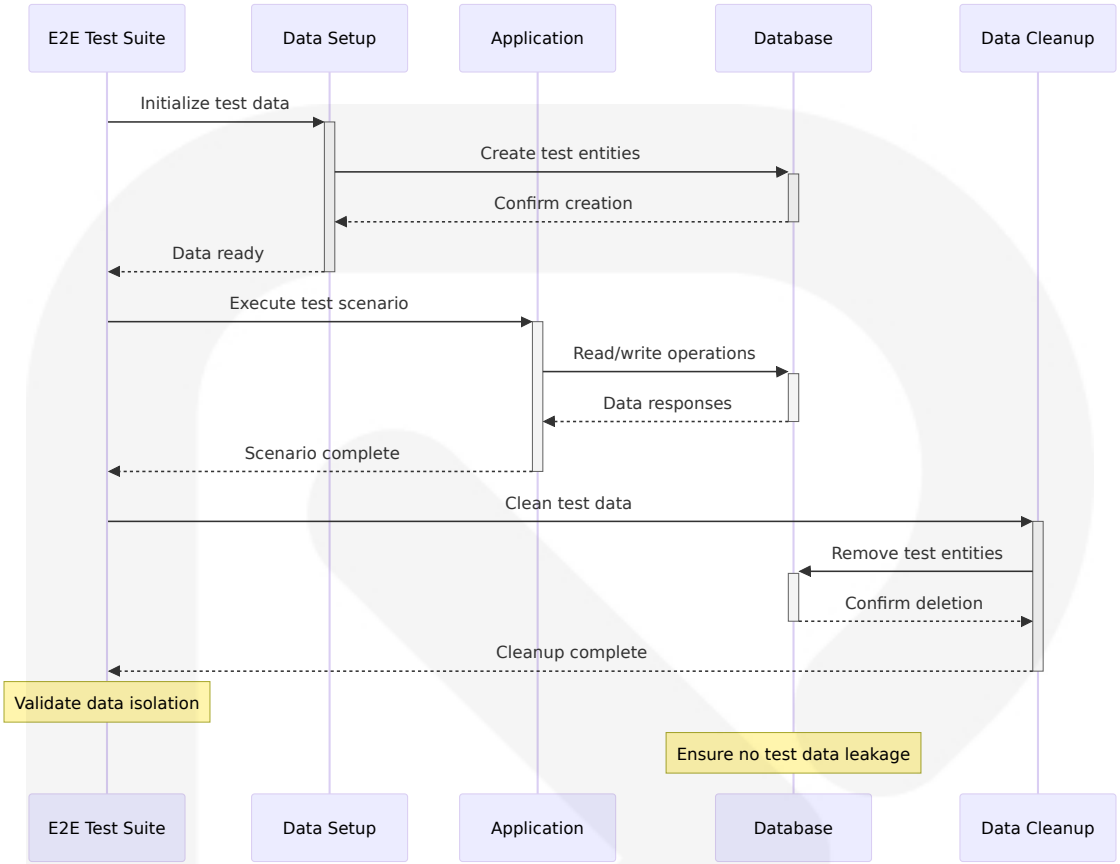
Isolated Test Data Management:

E2E tests utilize **isolated test data environments** with automated setup and teardown procedures to ensure test independence and data consistency across test runs.

Test Data Lifecycle Management:

Data Category	Setup Method	Teardown Method	Isolation Level	Performance Impact
User Accounts	Factory pattern creation	Automatic cleanup	Per test suite	<500ms setup
Asset Records	Seeded test assets	Database rollback	Per test case	<1s setup
KYC Documents	File upload simulation	File system cleanup	Per test run	<200ms setup
Compliance Rules	Rule engine seeding	Rule cache clearing	Per test suite	<300ms setup

Data Consistency Validation:



6.6.4.4 Performance Testing Requirements

E2E Performance Validation:

End-to-end performance testing validates that complete workflows meet the established SLA requirements while maintaining data integrity and security compliance throughout the entire user journey.

Performance Test Thresholds:

Workflow	Response Time Target	Throughput Target	Concurrency Level	Success Rate
User Authentication	<2s complete flow	100+ logins/min	50 concurrent users	99.5%
Asset Onboarding	<5min complete proces	10+ assets/hour	10 concurrent officers	99.0%

Workflow	Response Time Target	Throughput Target	Concurrency Level	Success Rate
	s			
KYC Verification	<30s verification	100+ verifications/hour	20 concurrent checks	98.0%
Compliance Export	<2min standard report	50+ exports/hour	5 concurrent exports	99.9%

6.6.4.5 Cross-Browser Testing Strategy

Comprehensive Browser Compatibility:

Cross-browser testing ensures compliance workflows function correctly across all supported browser environments, meeting accessibility standards and providing consistent user experience for regulatory users.

Browser Compatibility Matrix:

Browser	Version Support	Test Coverage	Specific Validations	Performance Baseline
Chrome	Latest + Previous 2	Full test suite	WebAuthn support, file uploads	100% baseline
Firefox	Latest + ESR	Core workflows	Cookie handling, CORS behavior	105% of baseline
Safari	Latest + Previous 1	Critical paths	iOS WebKit compatibility	110% of baseline
Edge	Latest + Previous 1	Core workflows	IE compatibility mode	102% of baseline

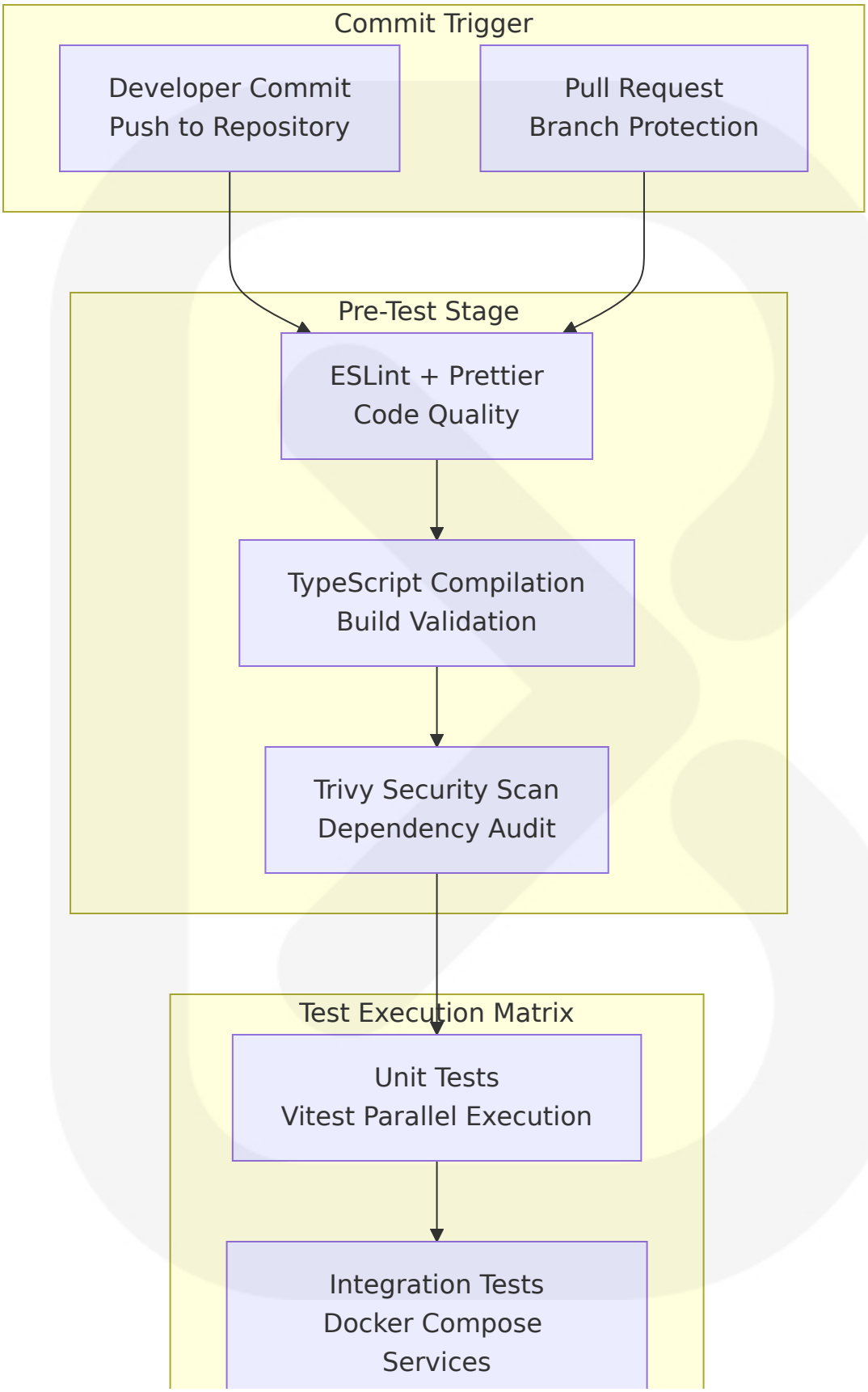
6.6.5 TEST AUTOMATION

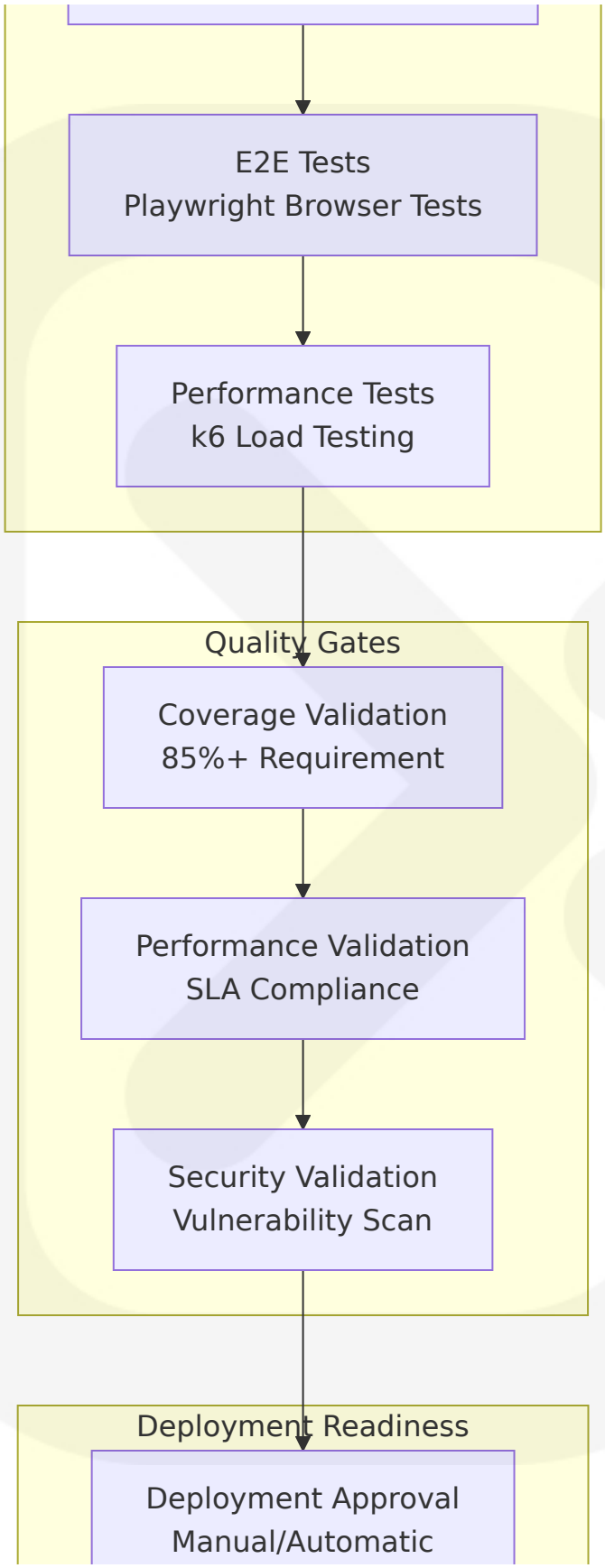
6.6.5.1 CI/CD Integration

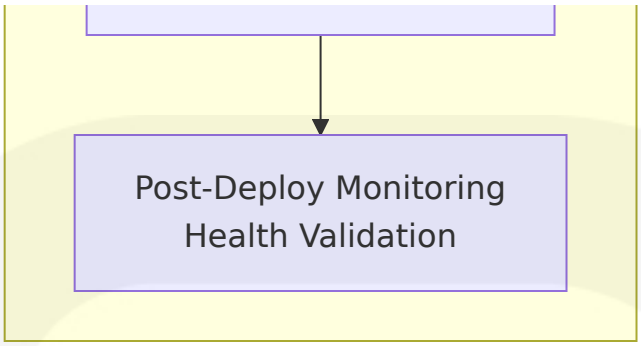
GitHub Actions Test Pipeline:

Veria implements a comprehensive **test automation pipeline** integrated with GitHub Actions, providing multi-stage test execution with parallel processing and intelligent failure handling for optimal developer productivity and deployment confidence.

CI/CD Test Pipeline Architecture:


Commit Trigger
Developer Commit
Push to Repository
Pull Request
Branch Protection
Pre-Test Stage
ESLint + Prettier
Code Quality
TypeScript Compilation
Build Validation
Trivy Security Scan
Dependency Audit
Test Execution Matrix
Unit Tests
Vitest Parallel Execution
Integration Tests
Docker Compose
Services


E2E Tests
Playwright Browser Tests
Performance Tests
k6 Load Testing
Quality Gates
Coverage Validation
85%+ Requirement
Performance Validation
SLA Compliance
Security Validation
Vulnerability Scan
Deployment Readiness
Deployment Approval
Manual/Automatic



Pipeline Configuration Standards:

Pipeline Stage	Execution Time Target	Failure Handling	Retry Policy	Notification Method
Code Quality	<2 minutes	Fast fail	No retry	Inline PR comments
Unit Tests	<5 minutes	Continue on non-critical	1 retry on timeout	Slack notification
Integration Tests	<10 minutes	Fail pipeline	2 retries on infrastructure	Email + Slack
E2E Tests	<15 minutes	Continue with warnings	1 retry on flaky	Detailed failure report
Performance Tests	<20 minutes	Fail on regression	No retry	Performance regression alert

6.6.5.2 Automated Test Triggers

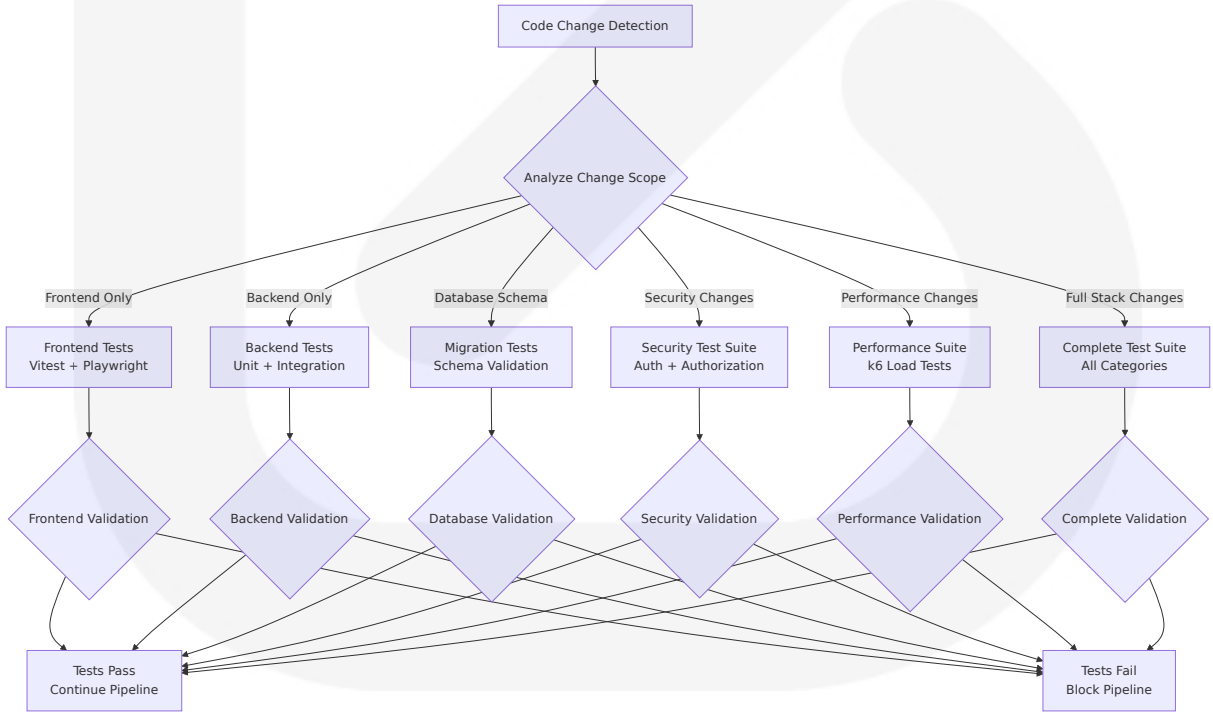
Multi-Trigger Test Automation:

The testing system implements **intelligent trigger mechanisms** that execute appropriate test suites based on change scope, branch type, and deployment target to optimize CI/CD pipeline efficiency while maintaining comprehensive coverage.

Trigger Configuration Matrix:

Trigger Event	Test Scope	Execution Environment	Performance Impact	Success Criteria
Feature Branch Push	Unit + Integration	Local Docker	<10 minutes	85%+ coverage
Pull Request	Full test suite	CI environment	<25 minutes	All tests pass
Main Branch Merge	Full + Performance	Staging environment	<35 minutes	SLA compliance
Release Tag	Complete validation	Production-like	<45 minutes	100% critical paths
Scheduled Nightly	Comprehensive + Security	Full environment	<60 minutes	Security compliance

Conditional Test Execution:



6.6.5.3 Parallel Test Execution

Optimized Test Parallelization:

Test execution utilizes **intelligent parallelization strategies** across multiple dimensions including test categories, service boundaries, and infrastructure resources to minimize total execution time while maintaining test isolation and reliability.

Parallelization Strategy:

Parallelization Level	Implementation	Resource Allocation	Isolation Method	Performance Gain
Test Category	Separate job matrices	4 parallel jobs	Independent runners	75% time reduction
Service Boundary	Per-service test suites	2 CPU cores per service	Docker containers	60% time reduction
Test Suite	Vitest worker threads	4 worker processes	Memory isolation	40% time reduction
Browser Tests	Playwright sharding	3 browser instances	Browser contexts	65% time reduction

Resource Management Matrix:

Resource Type	Allocation Strategy	Monitoring Method	Scaling Policy	Cost Optimization
CPU Cores	Dynamic allocation based on test type	Performance monitoring	Auto-scale to 8 cores max	Spot instances for non-critical
Memory	4GB base + 2GB per service	Memory usage tracking	Scale up to 16GB max	Memory-optimized instances
Storage	20GB SSD per runner	Disk usage monitoring	Temporary storage cleanup	Ephemeral storage

Resource Type	Allocation Strategy	Monitoring Method	Scaling Policy	Cost Optimization
Network	Dedicated test network	Bandwidth monitoring	QoS for critical tests	Shared network resources

6.6.5.4 Test Reporting Requirements

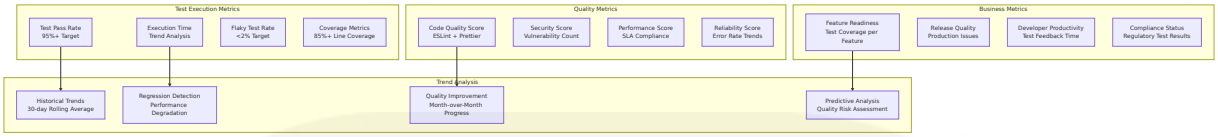
Comprehensive Test Reporting:

Test reporting provides **multi-stakeholder visibility** into test results, coverage metrics, performance trends, and quality gates with automated distribution and historical trend analysis for continuous improvement.

Report Distribution Matrix:

Stakeholder Group	Report Type	Delivery Method	Frequency	Content Focus
Developers	Detailed test results	GitHub PR comments	Per commit	Failure details, coverage changes
QA Team	Comprehensive test summary	Email + Slack	Per release	Test execution status, regression analysis
Product Management	Executive test summary	Dashboard + Email	Weekly	Feature readiness, quality metrics
Compliance Team	Security & compliance report	Secure email	Per deployment	Security test results, audit trail
DevOps Team	Infrastructure test metrics	Monitoring dashboard	Real-time	Performance metrics, resource usage

Test Metrics Dashboard:



6.6.5.5 Failed Test Handling

Intelligent Failure Management:

Failed test handling implements **automated triage and recovery mechanisms** with intelligent categorization of failure types, automatic retry policies for transient failures, and escalation procedures for persistent issues requiring human intervention.

Failure Classification System:

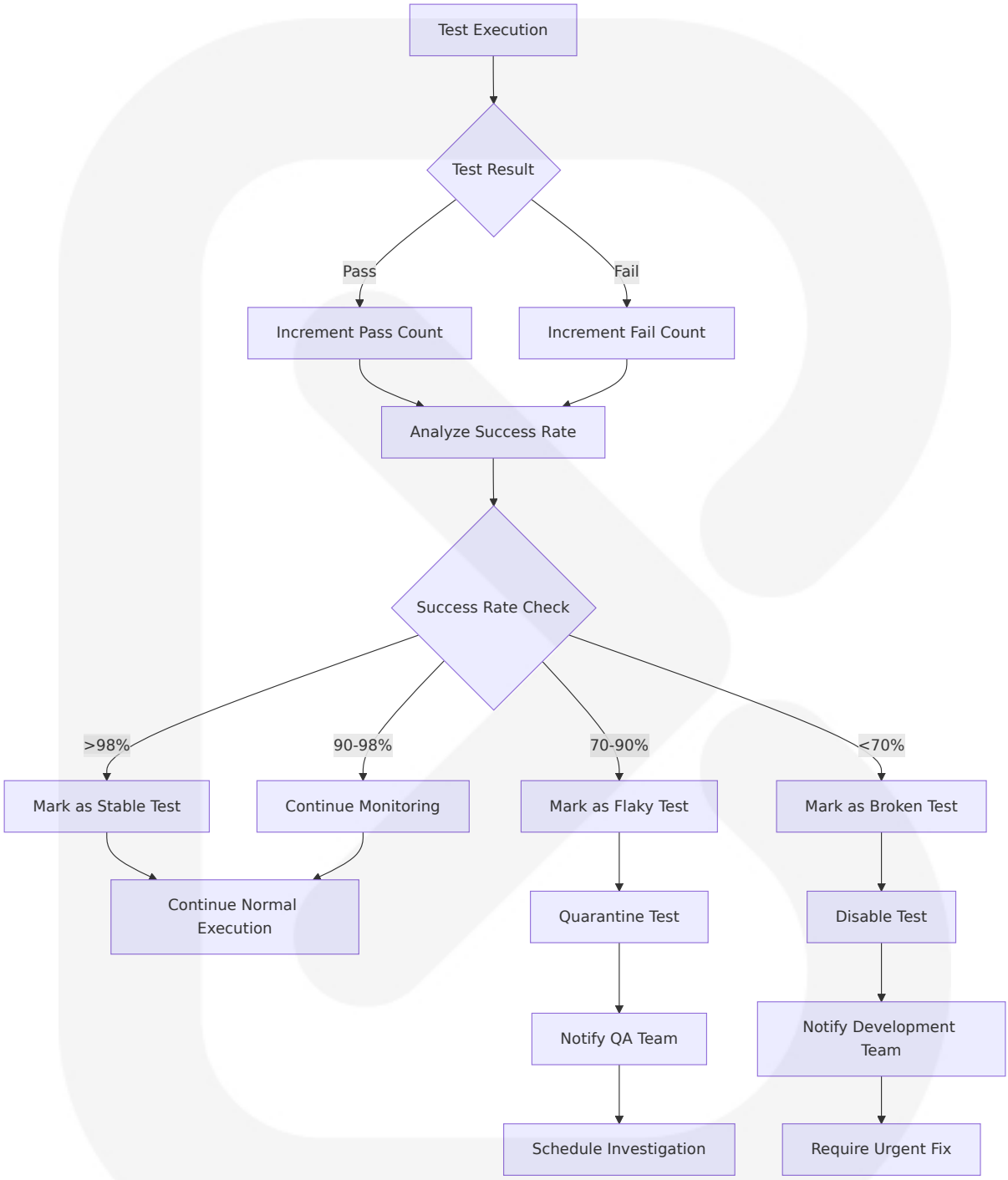
Failure Type	Automatic Recovery	Retry Policy	Escalation Time	Notification Level
Transient Infrastructure	Automatic retry	3 attempts with backoff	15 minutes	Developer notification
Flaky Tests	Mark as known flaky	2 attempts	30 minutes	QA team escalation
Regression Failures	Block deployment	No retry	Immediate	Senior developer alert
Performance Degradation	Continue with warning	1 retry	10 minutes	Performance team alert
Security Test Failures	Block deployment	No retry	Immediate	Security team alert

6.6.5.6 Flaky Test Management

Proactive Flaky Test Detection:

Flaky test management implements **statistical analysis and automated quarantine** for unreliable tests while maintaining overall pipeline stability and providing clear remediation paths for test reliability improvement.

Flaky Test Detection Algorithm:



Flaky Test Remediation Process:

Remediation Stage	Action	Timeline	Success Criteria	Escalation
Detection	Automatic quarantine	Immediate	Test isolation	QA notification
Analysis	Root cause investigation	48 hours	Failure pattern identified	Development team
Fix Implementation	Test stability improvement	5 business days	98%+ success rate	Engineering manager
Validation	Extended monitoring period	14 days	Stable test execution	Return to active suite

6.6.6 QUALITY METRICS

6.6.6.1 Code Coverage Targets

Comprehensive Coverage Requirements:

Code coverage targets align with regulatory compliance requirements and business risk assessment, implementing differentiated coverage thresholds based on component criticality and compliance impact.

Service-Specific Coverage Matrix:

Service Category	Line Coverage	Branch Coverage	Function Coverage	Critical Path Coverage	Justification
Identity Service	90%+	85%+	95%+	100%	Authentication security critical
Compliance Service	85%+	80%+	90%+	100%	Regulatory decision engine

Service Category	Line Coverage	Branch Coverage	Function Coverage	Critical Path Coverage	Justification
Gateway Service	85%+	80%+	90%+	100%	Entry point security
Audit Log Writer	95%+	90%+	100%	100%	Regulatory compliance critical
KYC Provider	80%+	75%+	85%+	95%	External integration dependency
Policy Service	90%+	85%+	95%+	100%	Business rule implementation
Shared Packages	90%+	85%+	95%+	100%	Cross-service dependencies

Coverage Exclusion Policies:

Exclusion Category	Rationale	Coverage Requirement	Alternative Validation
Configuration Files	Static configuration, no logic	None	Configuration validation tests
Type Definitions	TypeScript types, compile-time validation	None	Type checking in CI
Generated Code	Auto-generated, not business logic	None	Generator validation tests
Development Utilities	Non-production code	70%+	Manual testing acceptable

6.6.6.2 Test Success Rate Requirements

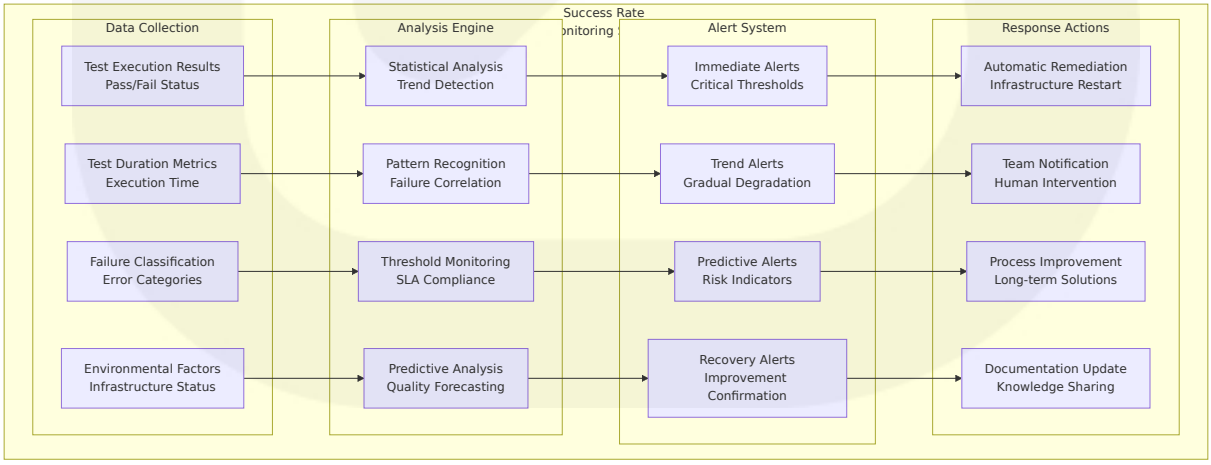
Reliability Standards:

Test success rate requirements establish **minimum reliability thresholds** for different test categories, ensuring pipeline stability while identifying areas requiring attention or improvement.

Success Rate Targets:

Test Category	Success Rate Target	Measurement Period	Failure Threshold	Action Required
Unit Tests	98%+	Rolling 7 days	<95%	Immediate investigation
Integration Tests	95%+	Rolling 7 days	<90%	Infrastructure review
E2E Tests	90%+	Rolling 7 days	<85%	User experience review
Performance Tests	95%+	Rolling 30 days	<90%	Performance optimization
Security Tests	99%+	Rolling 30 days	<98%	Security audit required

Success Rate Monitoring:



6.6.6.3 Performance Test Thresholds

SLA-Aligned Performance Validation:

Performance test thresholds directly validate the non-functional requirements established in section 2.5, ensuring system performance meets business SLA commitments under various load conditions.

Performance Threshold Matrix:

Performance Metric	Target Threshold	Warning Threshold	Critical Threshold	Test Methodology
API Response Time (p95)	<200ms	200-300ms	>300ms	k6 load testing with 1000 VU
Database Query Performance	<500ms	500-750ms	>750ms	Complex query benchmarking
Concurrent User Capacity	10,000+ users	8,000-10,000 users	<8,000 users	Sustained load testing
Transaction Throughput	10,000+ TPS	8,000-10,000 TPS	<8,000 TPS	Peak load simulation
System Memory Usage	<70% utilization	70-85% utilization	>85% utilization	Resource monitoring
CPU Utilization	<60% average	60-80% average	>80% average	Load pattern analysis

k6 Performance Test Configuration:

```
// Performance test thresholds aligned with SLA requirements
export let options = {
  thresholds: {
    'http_req_duration{p(95)}': ['<200'], // 95th percentile under 200ms
    'http_req_duration{p(99)}': ['<500'], // 99th percentile under 500ms
    'http_req_failed': ['<0.1%'],         // Error rate under 0.1%
```

```
    'http_reqs': ['rate>1000/s'],          // Throughput above 1000 RPS
    'iteration_duration': ['<2s'],          // Complete iterations under 2
  },
  stages: [
    { duration: '5m', target: 100 },      // Ramp up to 100 users
    { duration: '10m', target: 500 },     // Scale to 500 users
    { duration: '5m', target: 1000 },     // Peak load of 1000 users
    { duration: '10m', target: 1000 },    // Sustain peak load
    { duration: '5m', target: 0 },        // Ramp down
  ],
};
```

6.6.6.4 Quality Gates

Multi-Dimensional Quality Assessment:

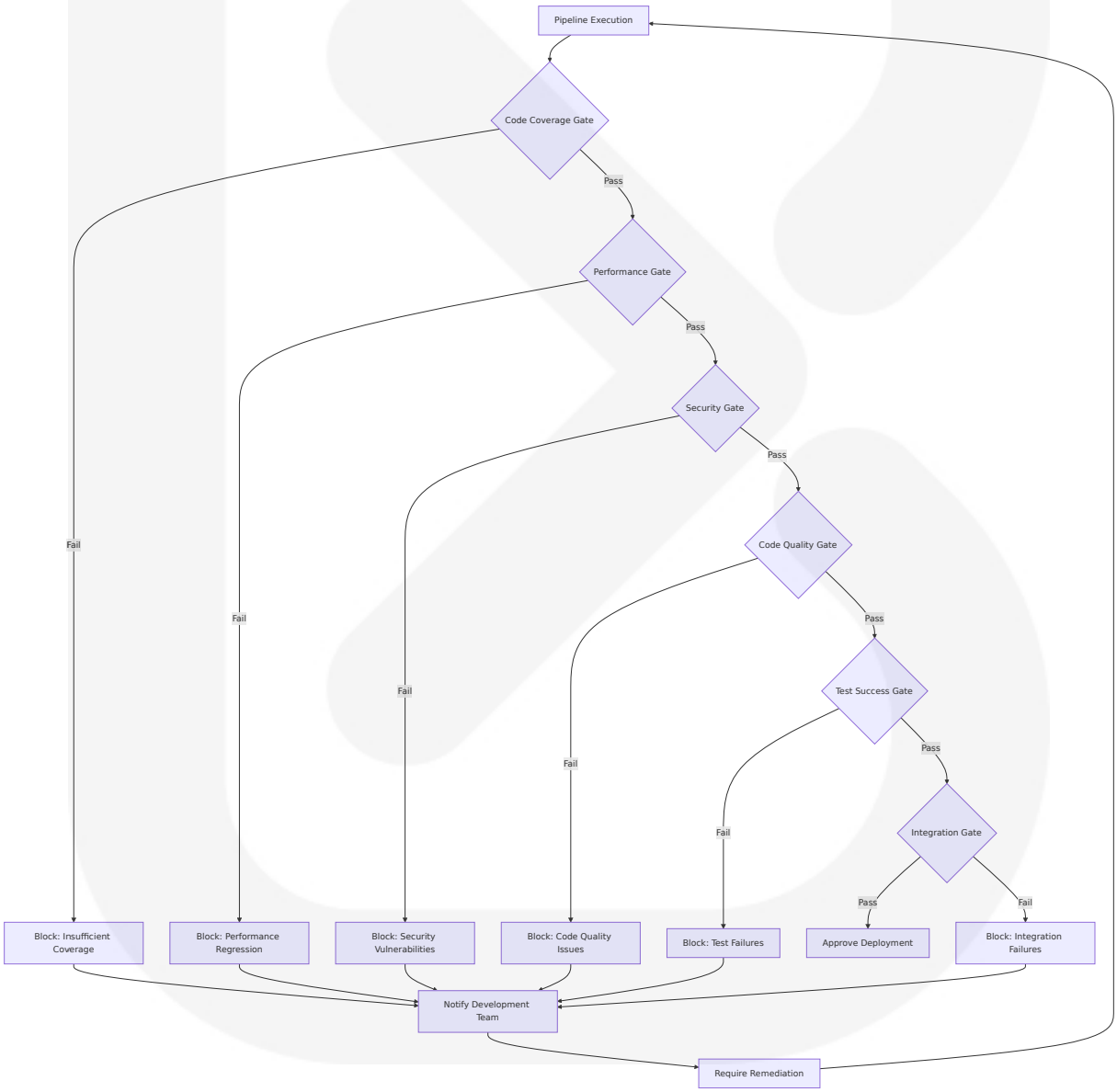
Quality gates implement **comprehensive quality criteria** that must be satisfied before code can proceed to the next stage of the deployment pipeline, ensuring both technical quality and business requirements are met.

Quality Gate Configuration:

Quality Gate	Criteria	Measurement Method	Pass Threshold	Block Deployment
Code Coverage Gate	Line + Branch coverage	Vitest coverage reports	85%+	Yes
Performance Gate	Response time compliance	k6 test results	<200ms p95	Yes
Security Gate	Vulnerability assessment	Trivy + pnpm audit	Zero high/critical	Yes
Code Quality Gate	ESLint + Prettier compliance	Static analysis	Zero errors	Yes

Quality Gate	Criteria	Measurement Method	Pass Threshold	Block Deployment
Test Success Gate	Test execution success	CI pipeline results	95%+ success rate	Yes
Integration Gate	Service compatibility	Contract testing	All contracts pass	Yes

Quality Gate Decision Matrix:



6.6.6.5 Documentation Requirements

Comprehensive Test Documentation:

Test documentation ensures **knowledge transfer**, **regulatory compliance**, and **maintenance efficiency** across the development lifecycle with standardized documentation patterns and automated generation where possible.

Documentation Standards Matrix:

Documentation Type	Scope	Update Frequency	Audience	Compliance Requirement
Test Plan Documentation	Overall testing strategy	Per release	QA + Development	Regulatory audit
Test Case Documentation	Individual test scenarios	Per feature	QA Team	Change management
Performance Test Reports	Load testing results	Per deployment	Operations + Management	SLA validation
Security Test Reports	Security validation results	Per release	Security + Compliance	Security audit
Test Environment Documentation	Environment configuration	Per environment change	DevOps + QA	Operational continuity

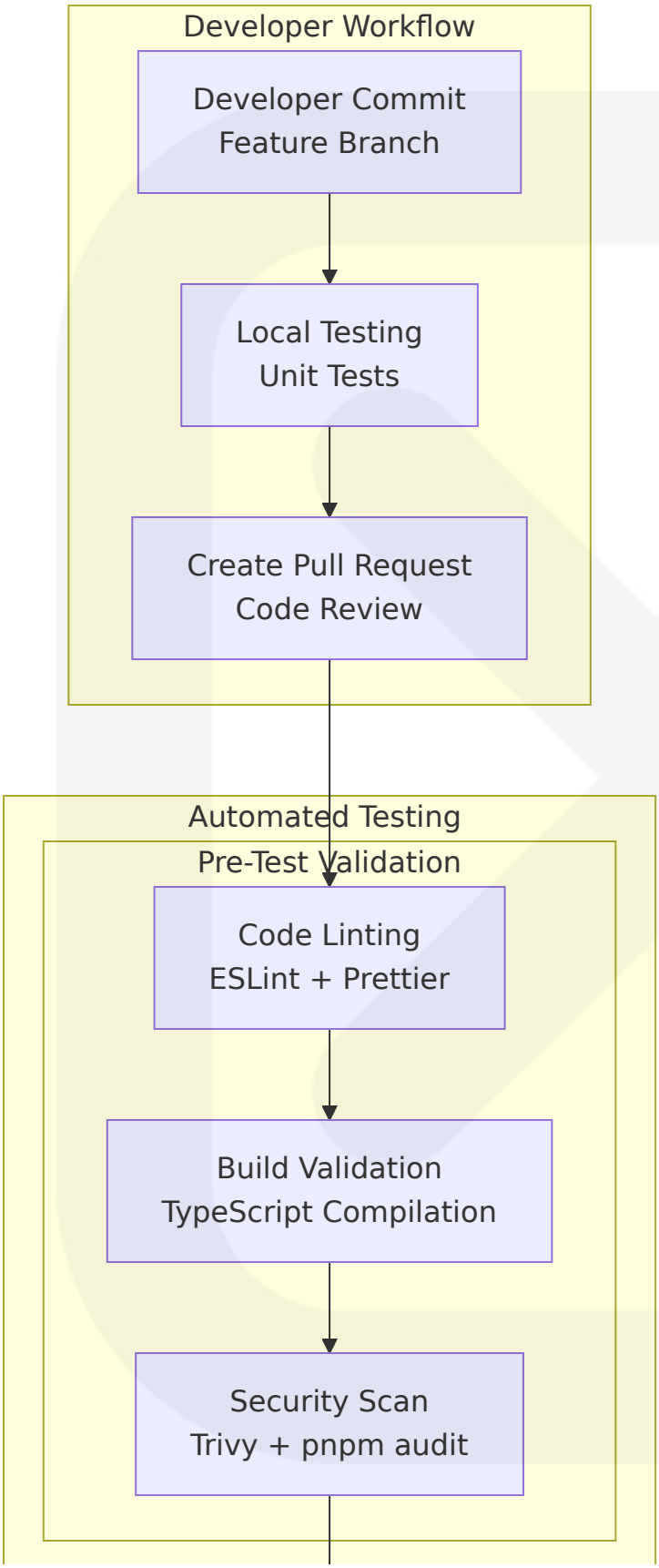
Automated Documentation Generation:

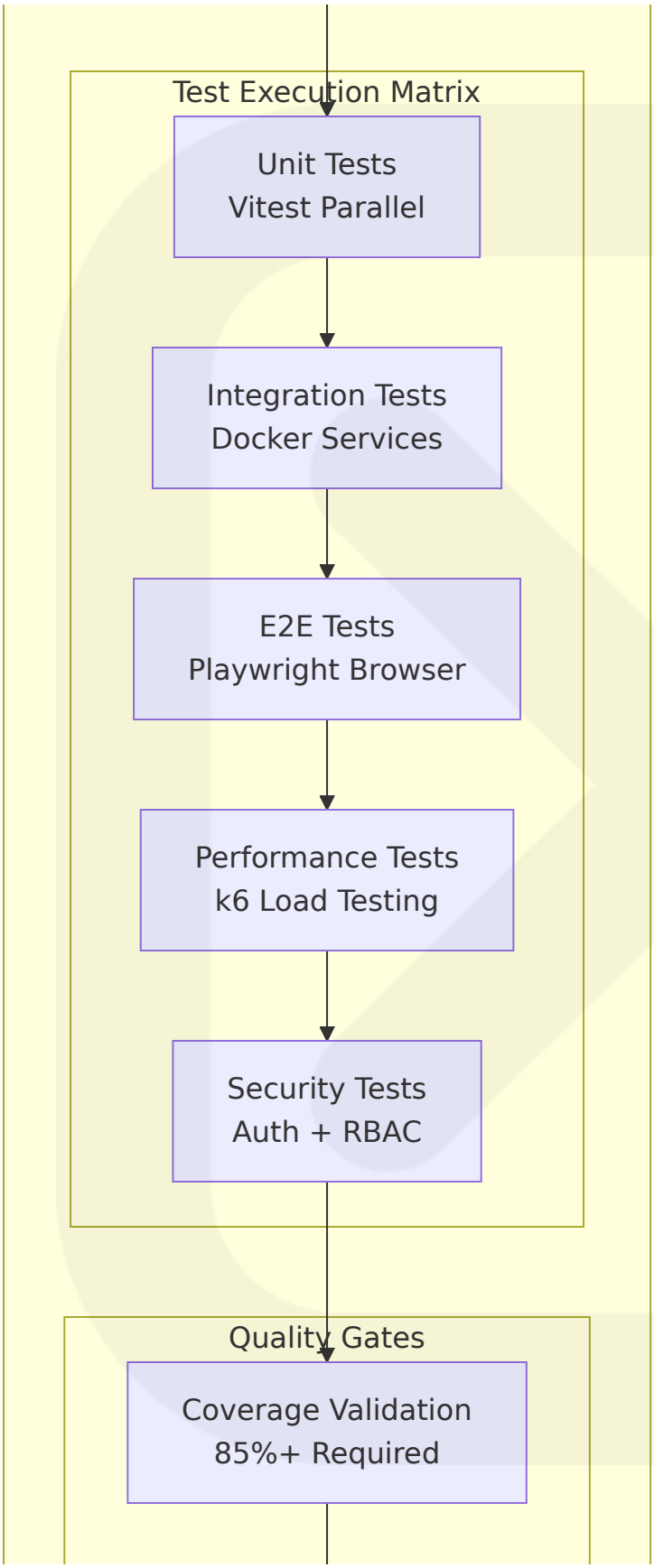
Generated Document	Source	Generation Trigger	Distribution Method	Retention Period
Test Coverage Reports	Vitest coverage data	Per CI run	Artifact upload + dashboard	90 days

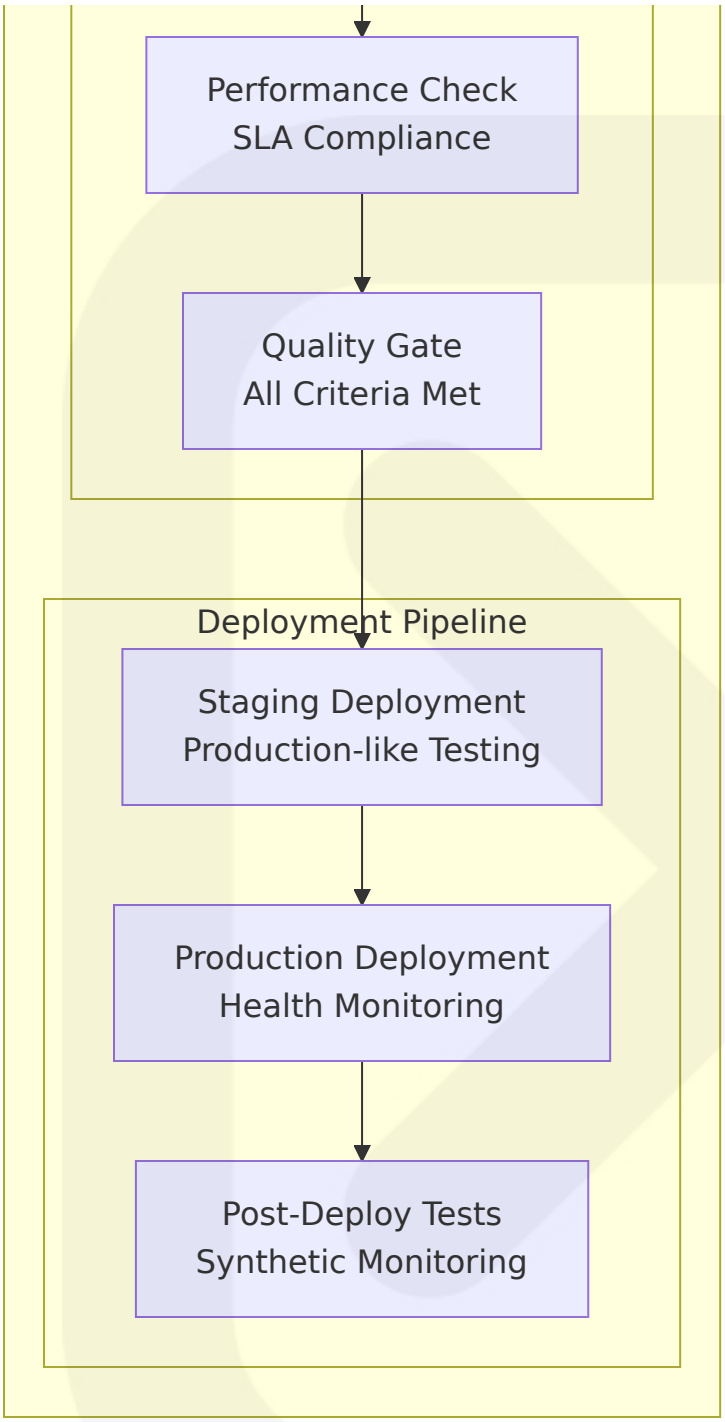
Generated Document	Source	Generation Trigger	Distribution Method	Retention Period
Performance Test Results	k6 execution data	Per performance test	S3 storage + email	1 year
API Test Documentation	OpenAPI + test results	Per API change	Documentation site	Current + 2 versions
Security Scan Reports	Trivy + audit results	Per security scan	Secure archive	2 years

6.6.7 TEST EXECUTION FLOW DIAGRAMS

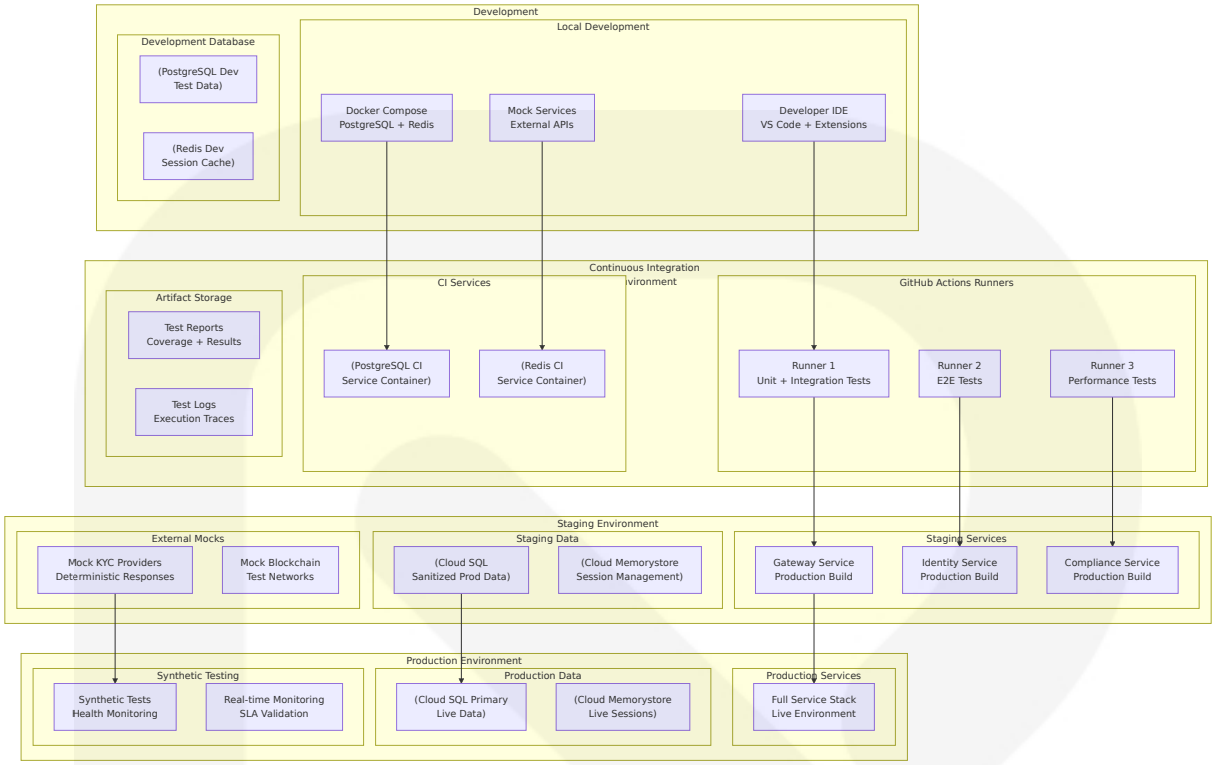
6.6.7.1 Complete Test Execution Flow


Developer Workflow
Developer Commit
Feature Branch
Local Testing
Unit Tests
Create Pull Request
Code Review
Automated Testing
Pre-Test Validation
Code Linting
ESLint + Prettier
Build Validation
TypeScript Compilation
Security Scan
Trivy + pnpm audit

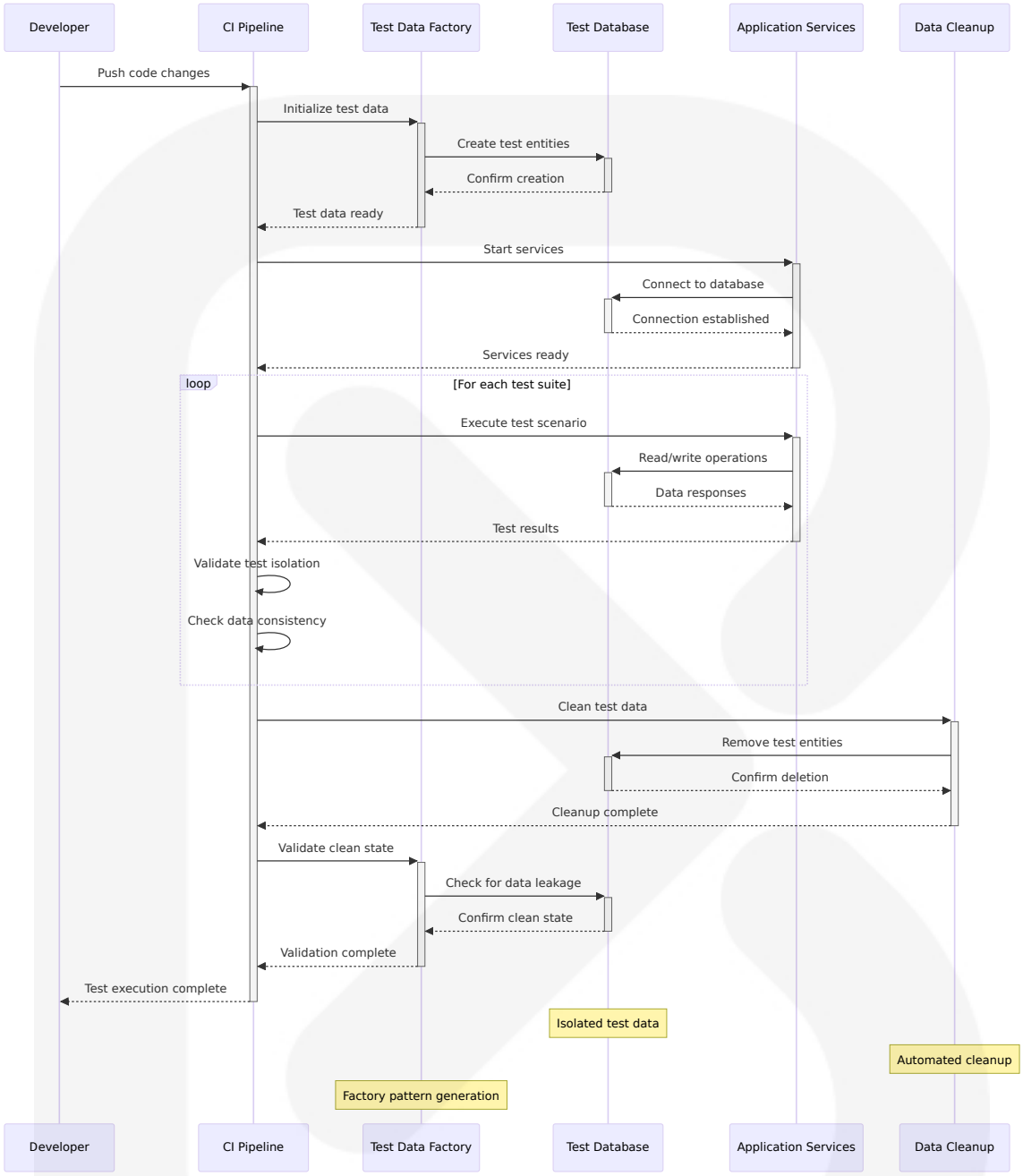

Test Execution Matrix
Unit Tests
Vitest Parallel
Integration Tests
Docker Services
E2E Tests
Playwright Browser
Performance Tests
k6 Load Testing
Security Tests
Auth + RBAC
Quality Gates
Coverage Validation
85%+ Required



6.6.7.2 Test Environment Architecture



6.6.7.3 Test Data Flow Architecture



6.6.8 SECURITY TESTING REQUIREMENTS

6.6.8.1 Authentication Security Testing

JWT Token Validation Testing:

Authentication security testing validates the complete JWT implementation including token generation, validation, refresh mechanisms, and blacklisting functionality to ensure robust authentication security.

Test Category	Test Scenarios	Security Validation	Expected Behavior
Token Generation	Valid credentials, invalid credentials	Cryptographic signature verification	Secure token creation or denial
Token Validation	Valid tokens, expired tokens, malformed tokens	Signature and expiry validation	Accept valid, reject invalid
Token Refresh	Valid refresh tokens, expired refresh tokens	Refresh token security	New tokens or secure denial
Token Blacklisting	Logout scenarios, security incidents	Redis blacklist functionality	Immediate token invalidation

WebAuthn Security Testing:

Multi-factor authentication testing validates the WebAuthn implementation including biometric authentication, hardware key support, and fallback mechanisms.

6.6.8.2 Authorization Security Testing

Role-Based Access Control (RBAC) Testing:

Authorization testing validates the comprehensive RBAC system with seven user roles and 50+ granular permissions to ensure proper access control enforcement across all system operations.

User Role	Permission Testing	Resource Access Testing	Escalation Testing
SUPER_ADMIN	Full permission validation	All resource access	Privilege escalation resistance

User Role	Permission Testing	Resource Access Testing	Escalation Testing
COMPLIANCE_OFFICER	KYC/AML permission testing	Compliance resource access	Horizontal privilege testing
INVESTOR	Limited permission validation	Self-service resource access	Privilege boundary testing
VIEWER	Read-only permission testing	Read-only resource access	Write operation denial

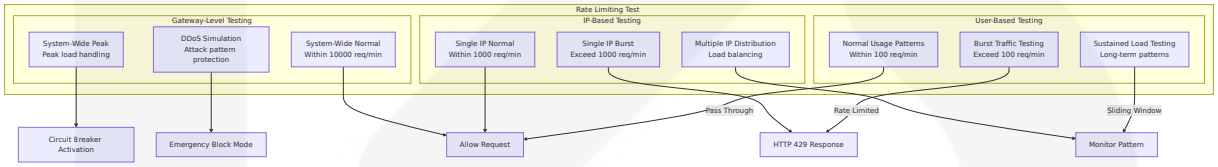
API-Level Authorization Testing:

Every API endpoint undergoes authorization testing to validate proper permission checking and access control enforcement.

6.6.8.3 Rate Limiting Security Testing

Multi-Tier Rate Limiting Validation:

Rate limiting testing validates the three-tier rate limiting system (user-based, IP-based, and gateway-level) to ensure proper protection against abuse and DoS attacks.



6.6.8.4 Audit Trail Security Testing

Comprehensive Audit Validation:

Audit trail testing validates the dual-write audit logging pattern to ensure complete compliance trail integrity and tamper-resistance for regulatory requirements.

Audit Component	Security Testing	Integrity Validation	Compliance Verification
Dual-Write Pattern	Write failure handling	Data consistency validation	Regulatory completeness
Audit Log Immutability	Tampering attempt detection	Hash verification	Regulatory integrity
Cross-Service Correlation	Request ID propagation	End-to-end traceability	Complete audit trail
Retention Policy	Automated cleanup testing	Data lifecycle management	Regulatory compliance

6.6.8.5 Data Protection Security Testing

Encryption and PII Protection:

Data protection testing validates AES-256 encryption implementation, PII masking, and secure data handling throughout the system.

Protection Layer	Test Scenarios	Validation Method	Security Standard
Data at Rest	Database encryption testing	Encryption verification	AES-256 compliance
Data in Transit	TLS 1.3 validation	Certificate and cipher testing	Transport security
PII Masking	Sensitive data handling	Masking pattern verification	Privacy protection
Key Management	Secret rotation testing	Key lifecycle validation	Cryptographic security

6.6.9 TEST ENVIRONMENT SPECIFICATIONS

6.6.9.1 Development Environment Requirements

Local Development Configuration:

Resource	Specification	Purpose	Performance Requirement
CPU	4+ cores, 2.5 GHz+	Parallel test execution	<30s unit test execution
Memory	16GB+ RAM	Service containers + IDE	<500MB per service
Storage	50GB+ SSD	Source code + containers	<100ms file I/O
Network	High-speed internet	Docker image pulls	<2Mbps sustained

6.6.9.2 CI Environment Specifications

GitHub Actions Runner Configuration:

Runner Type	Resource Allocation	Concurrent Tests	Usage Pattern
Standard Runner	2 CPU, 7GB RAM	2 parallel jobs	Unit + Integration tests
Large Runner	4 CPU, 16GB RAM	4 parallel jobs	E2E + Performance tests
GPU Runner	2 CPU + GPU, 14 GB RAM	1 specialized job	Future ML testing

6.6.9.3 Performance Testing Environment

Load Testing Infrastructure:

Performance testing requires dedicated infrastructure to generate realistic load patterns while measuring system performance under various conditions.

Load Generation	Specification	Capacity	Measurement Focus
k6 Runners	8 CPU, 32GB RAM	10,000+ virtual users	Throughput + latency
Database Load	Dedicated connection pool	1,000+ concurrent connections	Query performance
Network Simulation	Controlled latency/bandwidth	Various network conditions	Real-world performance

References

Repository Files Examined

- `package.json` - Root workspace test scripts and monorepo configuration
- `.github/workflows/ci.yml` - Complete CI/CD testing pipeline with GitHub Actions
- `services/gateway/vitest.config.ts` - Gateway service Vitest configuration with coverage settings
- `services/gateway/package.json` - Gateway service test dependencies and scripts
- `tests/e2e/playwright.config.ts` - Playwright end-to-end test configuration
- `contracts/package.json` - Smart contract testing configuration with Hardhat
- `packages/auth-middleware/vitest.config.ts` - Authentication middleware test configuration
- `tests/performance/load-test.js` - k6 performance testing implementation
- `docker-compose.yml` - Test environment service orchestration
- `.env.example` - Environment configuration template for testing

Repository Folders Explored

- `` (root) - Repository structure and monorepo test organization
- `tests/` - Central test directory with e2e, performance, and integration tests
- `tests/e2e/` - End-to-end test suites with Playwright and Vitest
- `.github/workflows/` - CI/CD automation and test execution workflows
- `services/` - Individual service test configurations and implementations
- `services/gateway/` - Gateway service unit and integration tests
- `packages/` - Shared package testing including auth middleware
- `apps/` - Frontend application testing with Vitest
- `apps/compliance-dashboard/` - React application testing configuration
- `contracts/` - Smart contract testing with Hardhat framework

Technical Specification Sections Referenced

- `2.5 NON-FUNCTIONAL REQUIREMENTS` - Performance targets and SLA requirements for test validation
- `3.6 DEVELOPMENT & DEPLOYMENT` - Testing framework ecosystem and CI/CD pipeline details
- `6.4 SECURITY ARCHITECTURE` - Security testing requirements for authentication, authorization, and audit trails
- `6.5 MONITORING AND OBSERVABILITY` - Performance monitoring and SLA requirements for test threshold alignment

7. USER INTERFACE DESIGN

7.1 FRONTEND APPLICATION ARCHITECTURE

7.1.1 Application Structure

The Veria platform implements a dual-frontend architecture optimized for different user personas and operational requirements:

7.1.1.1 Compliance Dashboard Application

- **Technology Stack**: React 18 + Vite + TypeScript
- **Development Port**: 3010
- **API Integration**: Proxy configuration routing `/api` â†’ `http://localhost:3001`
- **Primary Users**: Compliance officers, auditors, risk analysts
- **Completion Status**: ~10% complete with core components implemented
- **Purpose**: Real-time compliance monitoring, KYC verification management, and regulatory reporting

7.1.1.2 Main Frontend Application

- **Technology Stack**: Next.js 14 with App Router architecture
- **Environment Configuration**: NEXT_PUBLIC_GATEWAY_URL integration
- **Primary Users**: Investors, issuers, institutional clients
- **Completion Status**: ~5% complete with minimal implementation
- **Purpose**: Asset onboarding, investor management, treasury operations

7.1.2 Core UI Technology Stack

7.1.2.1 Frontend Frameworks and Build Systems

React v18 serves as the foundational UI framework, providing component-based architecture essential for the platform's complex form-heavy interfaces required for asset onboarding and investor management workflows. The declarative programming model aligns with compliance requirements for clear state management and comprehensive audit trails.

Next.js v14 implements the main frontend application using App Router architecture, enabling server-side rendering for improved initial load performance, static site generation for compliance documentation pages, and built-in API route handling for frontend-specific endpoints.

Vite powers the compliance dashboard build system with Hot Module Replacement (HMR) for rapid development cycles, optimized production builds with code splitting, and proxy configuration for seamless API integration during development.

7.1.2.2 Styling and Component Libraries

Tailwind CSS implements utility-first styling, chosen for consistency across large development teams and rapid prototype capabilities essential for compliance interface development. The framework provides comprehensive design system integration with custom color palette and responsive breakpoints.

Radix UI provides headless, accessible components ensuring WCAG compliance for institutional users, particularly important for enterprise clients with accessibility requirements. The library enables semantic HTML structure, keyboard navigation support, and screen reader compatibility.

Recharts handles data visualization for compliance reporting and analytics dashboards, supporting interactive charts with tooltips, responsive containers, and export capabilities for regulatory documentation.

Lucide React provides consistent iconography throughout the application with standardized visual language for status indicators, navigation elements, and action buttons.

7.1.2.3 State Management and Data Layer

@tanstack/react-query manages server state with intelligent caching, background updates, and optimistic updates for improved user experience

during compliance operations.

React hooks (useState, useEffect) handle local component state for form management, UI interactions, and real-time data display.

JWT Authentication implements dual-token pattern with 15-minute access tokens and 7-day refresh tokens, ensuring security while minimizing authentication overhead.

7.2 USER INTERFACE USE CASES

7.2.1 Asset Onboarding Workflow Interface

7.2.1.1 Asset Creation Flow

The asset onboarding interface guides compliance officers through the complete tokenization process:

1. **Asset Type Selection**: Dropdown interface for US Treasuries and Money Market Fund categories
2. **Custody Provider Configuration**: Integration forms for BNY Mellon API credentials with validation
3. **SPV/Trust Structure Setup**: Multi-step wizard for legal structure definition
4. **Tokenization Parameter Definition**: Forms for token symbol, supply limits, and decimal precision
5. **Regulatory Document Upload**: Drag-and-drop interface with hash integrity verification
6. **Compliance Rule Validation**: Real-time jurisdiction rule checking with approval workflow
7. **Smart Contract Deployment**: Progress tracking interface for ERC-3643 contract deployment

7.2.1.2 Document Management Interface

- File upload with format validation and virus scanning
- Version control interface for regulatory document updates
- Secure signed URL generation for document access
- Audit trail display for document changes and access events

7.2.2 Investor Management Workflow Interface

7.2.2.1 Registration and KYC Flow

The investor management interface handles the complete user lifecycle:

1. **User Registration**: Multi-step form with real-time validation and email verification
2. **Document Upload Interface**: KYC/KYB document submission with format validation
3. **Multi-Provider KYC Display**: Real-time status tracking across Chainalysis, TRM Labs, Jumio, and Onfido
4. **Risk Assessment Display**: Visual risk scoring with color-coded indicators
5. **Accreditation Verification**: Document upload and financial statement verification forms
6. **Approval Workflow**: Compliance officer review interface with approval/rejection actions

7.2.2.2 Status Tracking and Notifications

- Real-time verification status updates with progress indicators
- Email notification integration for status changes
- Dashboard widgets for pending approvals and verification queues

7.2.3 Compliance Export Workflow Interface

7.2.3.1 Audit Evidence Collection Interface

1. **Period Selection**: Date range picker with audit period validation
2. **Evidence Collection Progress**: Real-time progress tracking with completion indicators
3. **Export Generation**: ZIP file creation with manifest.json generation
4. **Download Interface**: Secure download links with 24-hour expiration
5. **Export History**: Complete audit trail of export activities with access timestamps

7.3 UI/BACKEND INTERACTION BOUNDARIES

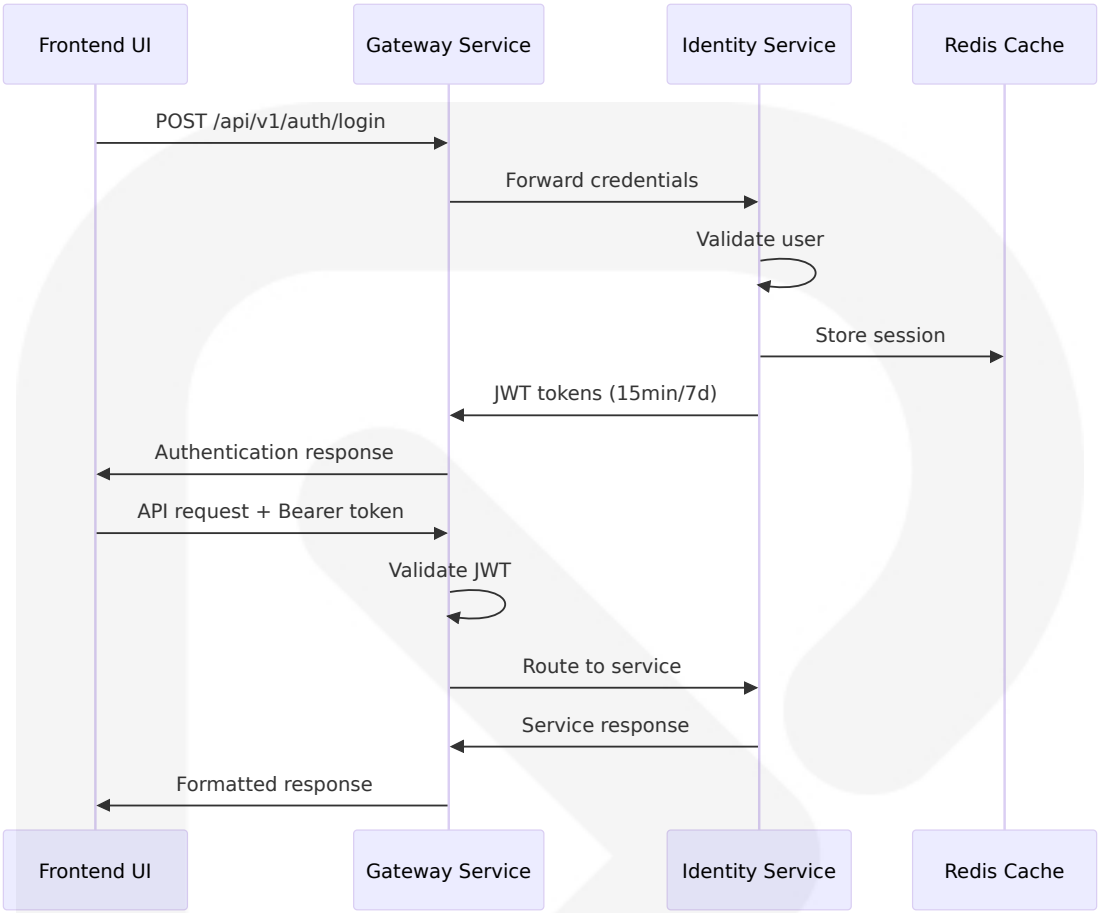
7.3.1 API Gateway Integration Pattern

7.3.1.1 Gateway Routing Architecture

All frontend applications communicate exclusively with the Gateway service on port 4000, which routes requests to appropriate backend services (ports 4001-4005). This pattern provides:

- **Centralized Authentication**: JWT token validation at gateway level
- **Rate Limiting**: 100 requests per 60-second window per IP address
- **Request/Response Transformation**: Consistent API format across services
- **Error Handling**: Standardized error response structure

7.3.1.2 Authentication Flow Integration



7.3.2 Service Communication Patterns

7.3.2.1 RESTful API Integration

Base URL Structure: `/api/v1/{service}/{resource}`
Authentication: Bearer token in Authorization header
Content-Type: `application/json` for all requests
Request ID Propagation: Automatic x-request-id header injection for distributed tracing

7.3.2.2 Key API Endpoints

- **Dashboard**: `GET /api/v1/dashboard/stats` - Real-time KPI data
- **KYC Management**: `GET /api/v1/kyc/verifications` - Verification status listing

- **Compliance**: `GET /api/v1/compliance/checks` - Compliance check results
- **Reporting**: `POST /api/v1/reports/generate` - Report generation triggers
- **Risk Analysis**: `GET /api/v1/risk/metrics` - Risk assessment data
- **Treasury**: `POST /api/treasury/deposit` - Treasury operations
- **User Management**: `POST /api/compliance/kyc` - KYC submission

7.3.3 Real-Time Data Integration

7.3.3.1 Auto-Refresh Patterns

- **Dashboard Components**: 30-second polling intervals for KPI updates
- **Compliance Checks**: 10-second refresh for real-time monitoring
- **Risk Analysis**: 30-second intervals for risk metric updates
- **Manual Refresh**: User-triggered refresh buttons for immediate updates

7.3.3.2 Development Mocking Strategy

The UI includes sophisticated development-time mocking:

- Axios interceptor returns deterministic mock data on network errors
- Comprehensive mock payloads for all major endpoints
- Automatic production build detection with mock disabling
- Enables UI development without backend dependencies

7.4 IMPLEMENTED SCREEN ARCHITECTURE

7.4.1 Dashboard Screen (`/`)

7.4.1.1 Key Performance Indicators Display

- **Total Users Card**: Real-time user count with trend indicators
- **KYC Verified Card**: Verification completion statistics
- **Pending Reviews Card**: Queue management for compliance officers
- **Risk Alerts Card**: High-priority risk notifications

7.4.1.2 Compliance Visualization Components

- **Compliance Trends Chart**: Multi-series line chart showing verified, pending, and rejected verification trends
- **Risk Heatmap**: Pie chart visualization with color-coded risk levels and percentage calculations
- **Recent Activity Feed**: Real-time activity stream with status icons and formatted timestamps

7.4.2 KYC Verifications Screen (`/kyc`)

7.4.2.1 Search and Filter Interface

- **Search Controls**: Real-time search across user names and identifiers
- **Status Filters**: Multi-select filtering by verification status
- **Risk Level Filters**: Color-coded risk level selection
- **Provider Status**: Integration status indicators for KYC providers

7.4.2.2 Verification Data Display

- **KYC Table Component**: Paginated table with loading, empty, and error states
- **Status Badges**: Color-coded status indicators with icons
- **Risk Level Display**: Progress bars and color coding for risk scores
- **Action Buttons**: Quick approval, rejection, and review actions

7.4.3 Compliance Checks Screen (`/compliance`)

7.4.3.1 Compliance Monitoring Interface

- **Compliance Check Listing**: Comprehensive check results display
- **Status Indicators**: PASSED/FAILED/WARNING badges with color coding
- **Score Visualization**: Progress bars for compliance scores
- **Flag Management**: Issue flag badges with severity indicators
- **Statistics Sidebar**: Aggregated compliance metrics and trends

7.4.4 Reports Screen (`/reports`)

7.4.4.1 Report Generation Interface

- **Report Type Filters**: Dropdown selection for SAR, CTR, compliance, and risk reports
- **Date Range Selection**: Calendar picker for report period definition
- **Quick Generate Buttons**: One-click report generation for common report types
- **Generation Status**: Real-time progress tracking with status indicators

7.4.4.2 Report Management

- **Report Listing**: Historical report display with timestamps and download links
- **Download Functionality**: Secure download with signed URLs
- **Export Status**: Generation progress and completion notifications

7.4.5 Risk Analysis Screen (`/risk`)

7.4.5.1 Risk Assessment Dashboard

- **Risk Score KPIs**: Current risk metrics with threshold indicators
- **Category Distribution**: Bar chart showing risk distribution across categories

- **Risk Indicators**: Traffic light system for risk thresholds
- **Recent Events Timeline**: Chronological risk event display with severity levels

7.4.6 Settings Screen (`/settings`)

7.4.6.1 Configuration Management Interface

- **Compliance Thresholds**: Configurable risk score limits and alert thresholds
- **Provider Settings**: Integration configuration for KYC providers (Chainalysis, TRM Labs, Jumio, Onfido)
- **Notification Preferences**: Email and system notification settings
- **Audit Log Retention**: Data retention policy configuration

7.5 USER INTERACTION PATTERNS

7.5.1 Navigation and Routing

7.5.1.1 Application Navigation

- **Sidebar Navigation**: Fixed left sidebar with icon-based navigation menu
- **Active State Highlighting**: Visual indication of current page with color coding
- **React Router Integration**: SPA navigation with history management
- **Breadcrumb Support**: Hierarchical navigation (planned implementation)

7.5.1.2 Navigation Items

- Dashboard: Overview and KPI monitoring
- KYC: Verification management and review

- Compliance: Rule checking and monitoring
- Reports: Regulatory report generation
- Risk: Risk analysis and monitoring
- Settings: System configuration

7.5.2 Form Interaction Patterns

7.5.2.1 Input Validation and Feedback

- **Real-time Validation**: Field-level validation with immediate feedback
- **Error State Display**: Red border and error message display
- **Success Indicators**: Green checkmarks for validated fields
- **Loading States**: Progress indicators during form submission

7.5.2.2 Form Submission Workflows

- **Controlled Components**: React state management for all form inputs
- **Validation Schema**: Zod-based validation for data integrity
- **Error Handling**: Comprehensive error display with recovery options
- **Success Confirmation**: Modal or toast notifications for successful submissions

7.5.3 Data Visualization Interactions

7.5.3.1 Chart Interactions

- **Interactive Tooltips**: Hover states with detailed data display
- **Responsive Design**: Chart scaling for different screen sizes
- **Legend Integration**: Interactive legend with data series toggling
- **Export Capabilities**: Download charts as images or PDF (planned)

7.5.3.2 Table Interactions

- **Sorting**: Column-based sorting with visual indicators
- **Pagination**: Page-based navigation for large datasets

- **Selection**: Row selection for batch operations
- **Filtering**: Column-level filtering with search inputs

7.6 VISUAL DESIGN SYSTEM

7.6.1 Color Palette and Theming

7.6.1.1 Primary Color System

- **Primary Colors**: 50-900 shade system defined in Tailwind configuration
- **Status Colors**:
  - Success/Verified/Low Risk: Green (#10b981)
  - Warning/Pending/Medium Risk: Yellow (#f59e0b)
  - Error/Failed/High Risk: Red (#ef4444)
  - Critical Risk: Dark Red (#991b1b)

7.6.1.2 Accessibility Compliance

- **WCAG 2.1 AA Compliance**: Color contrast ratios meet accessibility standards
- **Color-blind Friendly**: Alternative indicators for color-dependent information
- **High Contrast Mode**: Support for system high contrast preferences

7.6.2 Typography and Layout

7.6.2.1 Typography System

- **Font Family**: System font stack with Tailwind CSS defaults
- **Heading Hierarchy**: Consistent h1-h6 sizing and spacing
- **Body Text**: Optimized line height and letter spacing for readability
- **Code Text**: Monospace font for technical identifiers and codes

7.6.2.2 Layout Patterns

- **Responsive Grid**: 1-4 column grid layouts with breakpoints
- **Card-based Design**: Consistent card component structure with shadows
- **Spacing System**: 8px base unit with 4px, 8px, 16px, 24px, 32px increments
- **Breakpoints**: Mobile-first responsive design with md (768px) and lg (1024px) breakpoints

7.6.3 Component Design Standards

7.6.3.1 Button System

- **Primary Actions**: Blue background with white text
- **Secondary Actions**: Gray border with dark text
- **Danger Actions**: Red background for destructive operations
- **Disabled States**: Reduced opacity with cursor restrictions

7.6.3.2 Status Indicators

- **Badge Components**: Rounded badges with appropriate color coding
- **Progress Bars**: Animated progress indicators with percentage display
- **Loading Spinners**: Consistent loading animation across components
- **Icon Integration**: Lucide React icons with consistent sizing and styling

7.7 COMPONENT LIBRARY ARCHITECTURE

7.7.1 Core UI Components

7.7.1.1 Layout Components

Layout Component serves as the primary application shell with:

- Fixed left sidebar navigation with collapsible functionality
- Main content area with router outlet for page rendering
- Responsive header with user profile and notification areas
- Footer with system status and version information

7.7.1.2 Data Display Components

StatsCard Component provides standardized KPI display with:

- Title, value, and change indicator formatting
- Icon integration with consistent sizing
- Trend arrows (up/down/neutral) with color coding
- Background color variations based on data type

ComplianceChart Component implements multi-series visualization:

- Line chart with three data series (verified, pending, rejected)
- Responsive container with automatic scaling
- Interactive legend with series toggling
- Tooltip display with formatted data points

KycTable Component handles verification data display:

- Loading, empty, and error state management
- Status badges with icons and color coding
- Risk level visualization with progress bars
- Sortable columns with pagination support
- Action button integration for workflow operations

RiskHeatmap Component provides risk assessment visualization:

- Pie chart with color-coded risk level segments
- Percentage calculations with automatic formatting
- Interactive tooltips with detailed risk information
- Legend display with risk category definitions

7.7.2 Form and Input Components

7.7.2.1 Input Components

- **Text Inputs**: Styled input fields with validation states
- **Select Dropdowns**: Custom select components with search functionality
- **File Upload**: Drag-and-drop interface with progress indicators
- **Date Pickers**: Calendar integration for date range selection
- **Multi-Select**: Checkbox-based multiple selection interface

7.7.2.2 Validation Components

- **Error Display**: Consistent error message formatting
- **Field Validation**: Real-time validation with visual feedback
- **Form State Management**: Loading and success states
- **Required Field Indicators**: Visual marking for mandatory fields

7.8 MISSING UI IMPLEMENTATION REQUIREMENTS

7.8.1 Asset Tokenization Interface Requirements

7.8.1.1 Asset Creation Workflow UI

Based on the feature catalog, the following interfaces require implementation:

- **Asset Creation Forms**: Multi-step wizard for asset parameter definition
- **Document Upload Interface**: Regulatory document management with version control

- **Tokenization Configuration**: Smart contract parameter configuration forms
- **Deployment Status Monitoring**: Real-time contract deployment progress tracking

7.8.1.2 Custody Provider Integration UI

- **Provider Configuration**: API credential management interface
- **Connection Testing**: Real-time API connectivity verification
- **Fallback Configuration**: Multiple provider setup with priority ordering

7.8.2 Treasury Operations Dashboard Requirements

7.8.2.1 Transaction Management Interface

- **Deposit/Withdrawal Forms**: Multi-signature transaction creation
- **Balance Displays**: Real-time balance tracking across multiple assets
- **Transaction History**: Comprehensive transaction log with filtering
- **Approval Workflows**: Multi-signature approval queue management

7.8.2.2 Treasury Analytics Interface

- **Cash Flow Analysis**: Historical cash flow visualization
- **Yield Tracking**: Asset performance metrics and reporting
- **Reconciliation Dashboard**: Automated reconciliation status and discrepancy reporting

7.8.3 Advanced User Management Interface Requirements

7.8.3.1 Organization Management UI

- **Organization Profiles**: Comprehensive organization information management
- **KYB Status Tracking**: Business verification status and document management
- **User-Organization Relationships**: Role assignment and permission management
- **Multi-Tenant Access Control**: Organization-scoped data access interface

7.8.3.2 Advanced Authentication Interface

- **WebAuthn Integration**: Biometric and hardware key authentication setup
- **Multi-Factor Authentication**: TOTP and SMS-based 2FA configuration
- **Session Management**: Active session display and remote logout capabilities

7.8.4 Smart Contract Management Interface Requirements

7.8.4.1 Contract Deployment Wizard

- **Parameter Configuration**: Token specification and compliance rule setup
- **Deployment Monitoring**: Real-time deployment status and gas fee tracking
- **Contract Verification**: Automated contract verification and audit integration
- **Upgrade Management**: Contract upgrade workflow and approval process

7.9 DEVELOPMENT AND DEPLOYMENT

CONSIDERATIONS

7.9.1 Development Environment Setup

7.9.1.1 Local Development Configuration

- **Port Configuration**: Compliance Dashboard (3010), API Gateway (4000)
- **API Proxy**: Automatic routing to backend services during development
- **Hot Module Replacement**: Real-time UI updates during development
- **Mock Data Integration**: Development-time data mocking for independent UI development

7.9.1.2 Build and Deployment

- **Production Builds**: Optimized builds with code splitting and minification
- **Environment Variables**: Runtime configuration for API endpoints and feature flags
- **Static Asset Optimization**: Image optimization and CDN integration
- **Progressive Web App**: Service worker implementation for offline functionality (planned)

7.9.2 Performance Optimization

7.9.2.1 Frontend Performance

- **Code Splitting**: Route-based code splitting for optimal loading
- **Lazy Loading**: Component lazy loading for improved initial load times
- **Caching Strategy**: React Query caching with appropriate TTL settings
- **Bundle Size Optimization**: Tree shaking and dependency optimization

7.9.2.2 User Experience Optimization

- **Loading States**: Consistent loading indicators across all components
- **Error Boundaries**: Graceful error handling with recovery options
- **Accessibility Features**: Keyboard navigation and screen reader support
- **Mobile Responsiveness**: Touch-friendly interface design for tablet and mobile use

References

Technical Specification Sections Referenced

- `1.2 SYSTEM OVERVIEW` - System architecture and integration patterns
- `2.1 FEATURE CATALOG` - Complete feature requirements and UI needs
- `3.2 FRAMEWORKS & LIBRARIES` - Frontend technology stack details
- `4.1 SYSTEM WORKFLOWS` - User interaction workflows and UI flow requirements
- `5.2 COMPONENT DETAILS` - Backend service integration and API patterns

Files and Folders Examined

- `apps/compliance-dashboard/src/App.tsx` - React application routing and structure
- `apps/compliance-dashboard/src/services/api.ts` - Frontend API integration patterns
- `apps/compliance-dashboard/package.json` - Frontend dependencies and build configuration
- `apps/compliance-dashboard/tailwind.config.js` - Design system configuration
- `apps/compliance-dashboard/vite.config.ts` - Build system and development server setup
- `apps/compliance-dashboard/src/components/` - UI component library implementation
- `apps/compliance-dashboard/src/pages/` - Screen-level component architecture

- `apps/frontend/app/products/page.tsx` - Next.js application structure
- `docs/API.md` - Backend API endpoint specifications
- `docs/ARCHITECTURE.md` - System architecture overview
- `docs/PRODUCT_REQUIREMENTS.md` - Product vision and requirements

8. INFRASTRUCTURE

8.1 DEPLOYMENT ENVIRONMENT

8.1.1 Target Environment Assessment

8.1.1.1 Environment Type Classification

Veria implements a **hybrid multi-cloud architecture** designed specifically for financial compliance middleware requirements. The platform combines cloud-native scalability with regulatory compliance standards, utilizing both Google Cloud Platform and Amazon Web Services to ensure high availability and geographic redundancy.

Primary Cloud Strategy: The architecture leverages Google Cloud Run for serverless container deployments combined with AWS EKS for Kubernetes orchestration, providing flexibility in deployment models based on specific operational requirements and cost optimization strategies.

Geographic Distribution Requirements: The platform maintains production deployments across multiple regions to support compliance with data sovereignty requirements:

- **Primary Region**: us-central1 (Google Cloud) / us-east-1 (AWS)
- **Secondary Region**: us-east1 (warm standby)
- **Disaster Recovery**: europe-west1 (cold standby for EU compliance)

8.1.1.2 Resource Requirements

Compute Resource Allocation:

Service Category	CPU Requirements	Memory Requirements	Scaling Profile
Gateway Service	1-2 cores	512Mi-1Gi	High request volume, low compute
Identity Service	0.5-1 core	512Mi	JWT operations, session management
Compliance Service	2-4 cores	2-4Gi	Complex rule evaluation, high compute
KYC Provider Service	1-2 cores	1-2Gi	I/O bound external integrations
Blockchain Service	1-2 cores	1Gi	Network-dependent operations

Storage Requirements:

- **Database Storage**: PostgreSQL with 100GB initial allocation, auto-scaling to 1TB
- **Redis Cache**: 4GB memory allocation with data persistence enabled
- **Audit Logs**: 500GB initial allocation with 7-year retention policy
- **Container Registry**: Multi-registry storage (GHCR, GAR, ECR) for deployment redundancy

Network Requirements:

- **Bandwidth**: 1Gbps minimum per service instance for external API integrations
- **Latency**: <5ms inter-service communication within the same region
- **Concurrent Connections**: 10,000+ concurrent API connections per service cluster
- **External Integrations**: Dedicated bandwidth for KYC providers and blockchain networks

8.1.1.3 Compliance and Regulatory Requirements

Financial Services Compliance: The infrastructure supports multiple regulatory frameworks:

- **SOX Compliance**: Immutable audit trails with 7-year retention
- **AML/KYC Compliance**: Real-time screening and monitoring capabilities
- **SEC Regulations**: Comprehensive reporting and data export capabilities
- **GDPR Compliance**: EU data residency and privacy controls

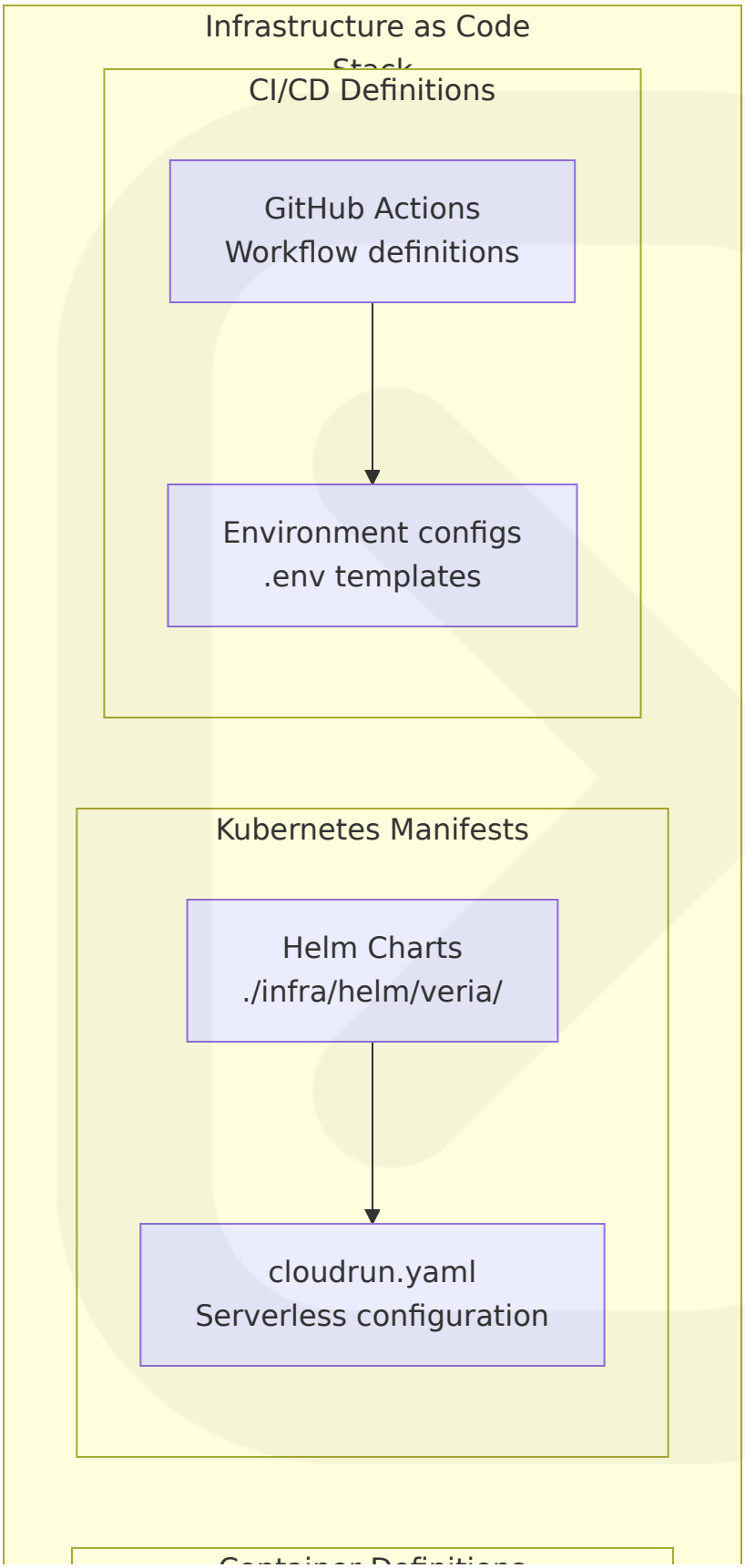
Security Standards: Infrastructure implements enterprise-grade security:

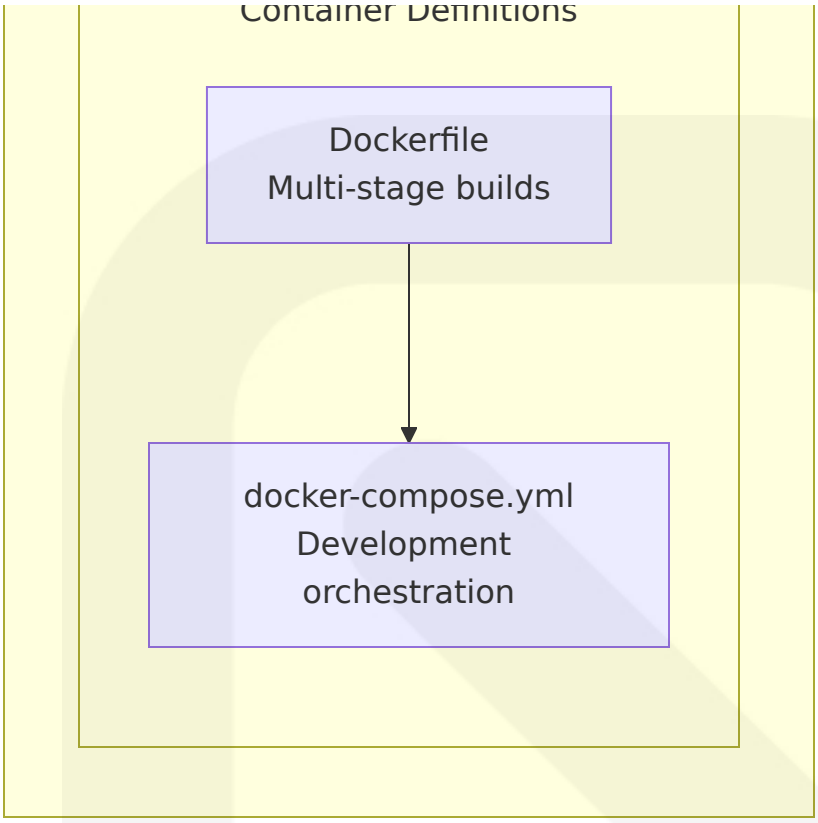
- **Data Encryption**: AES-256 at rest, TLS 1.3 in transit
- **Access Controls**: Multi-factor authentication with WebAuthn support
- **Network Security**: Zero-trust architecture with service mesh isolation
- **Audit Requirements**: Comprehensive logging with tamper-evident storage

8.1.2 Environment Management

8.1.2.1 Infrastructure as Code Approach

Containerized Infrastructure: The platform utilizes a Docker-first approach with Infrastructure as Code patterns:


Infrastructure as Code
CI/CD Definitions
GitHub Actions
Workflow definitions
Environment configs
.env templates
Kubernetes Manifests
Helm Charts
./infra/helm/veria/
cloudrun.yaml
Serverless configuration



Configuration Management Strategy: The system implements environment-specific configuration through:

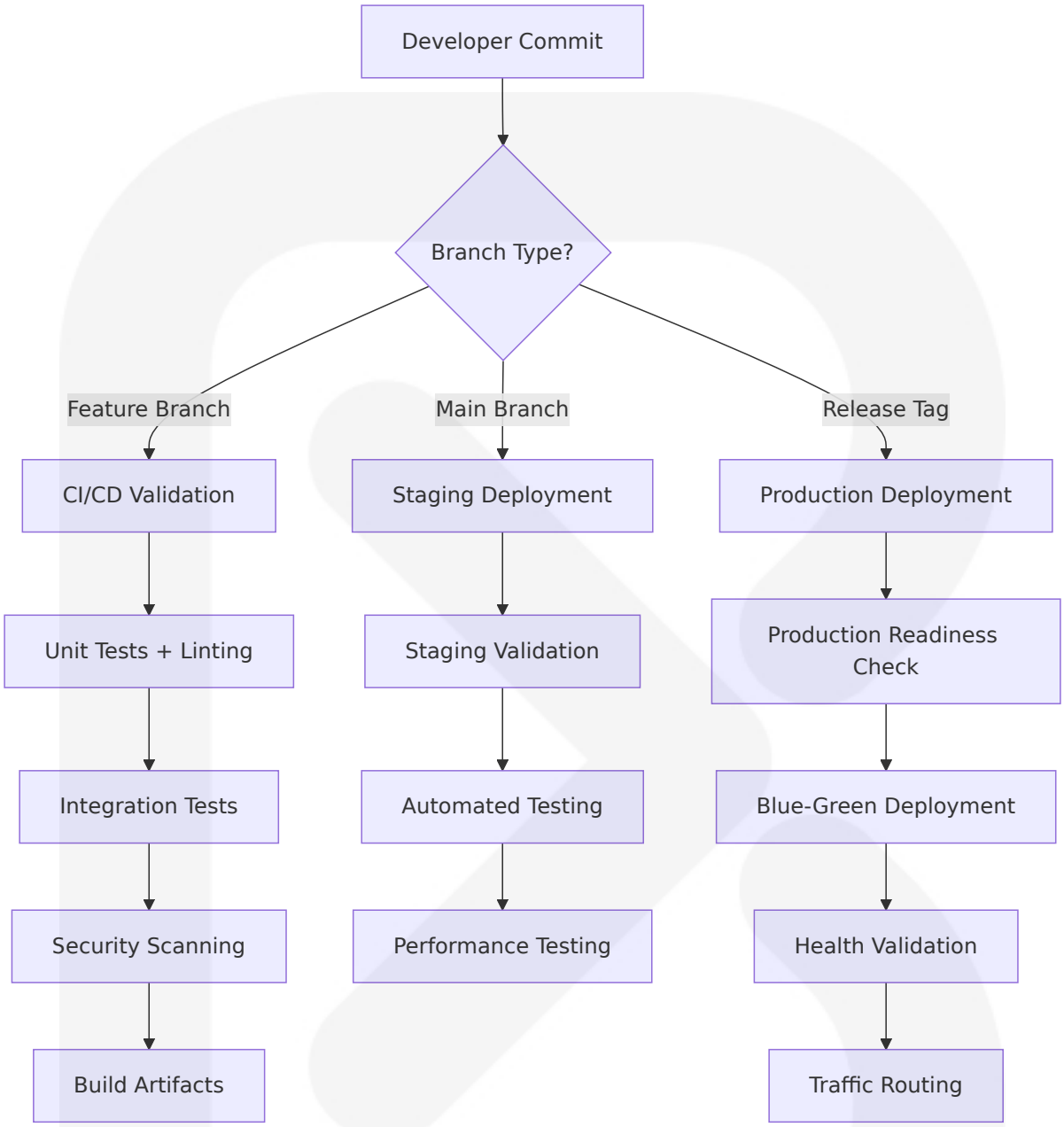
- **Environment Variables**: Runtime configuration injection via cloud secret managers
- **Docker Compose**: Local development environment standardization
- **Helm Charts**: Kubernetes deployment templating and configuration management
- **Cloud Run YAML**: Serverless container configuration with auto-scaling parameters

8.1.2.2 Environment Promotion Strategy

Three-Tier Environment Architecture:

Environment	Purpose	Deployment Trigger	Configuration
Development	Local development and testing	Manual via Docker Compose	Single replicas, mock services
Staging	Integration testing and QA	Automatic on main branch merge	Production-like with reduced capacity
Production	Live system operations	Tag-based deployment (v*)	Full redundancy with auto-scaling

Promotion Workflow:



8.1.2.3 Backup and Disaster Recovery Plans

Database Backup Strategy:

- **Continuous Backups**: PostgreSQL WAL shipping with 5-minute intervals
- **Point-in-Time Recovery**: 30-day recovery window with hour-level granularity

- **Cross-Region Replication**: Asynchronous replication to secondary regions
- **Backup Verification**: Automated restore testing on backup snapshots

Application State Backup:

- **Configuration Backups**: Version-controlled infrastructure configurations
- **Secret Management**: Encrypted backup of secrets and certificates
- **Container Images**: Multi-registry storage for deployment artifact redundancy
- **Audit Log Preservation**: Immutable storage with geographic distribution

Recovery Time Objectives:

Component	RTO (Recovery Time)	RPO (Recovery Point)	Recovery Method
Database Recovery	<30 minutes	<5 minutes	Automated failover
Service Instances	<5 minutes	Immediate	Auto-scaling replacement
Complete System	<60 minutes	<15 minutes	Multi-region failover
Data Center Loss	<4 hours	<30 minutes	Cross-region promotion

8.2 CLOUD SERVICES

8.2.1 Cloud Provider Selection and Justification

8.2.1.1 Multi-Cloud Architecture Rationale

Google Cloud Platform (Primary):

- **Justification**: Superior serverless container platform (Cloud Run) with automatic scaling and cost optimization
- **Core Services**: Cloud Run, Artifact Registry, Secret Manager, Cloud SQL
- **Cost Efficiency**: Pay-per-use pricing model ideal for variable compliance workloads
- **Integration**: Native Kubernetes integration and advanced AI/ML services for future expansion

Amazon Web Services (Secondary):

- **Justification**: Enterprise-grade EKS for complex orchestration requirements and regulatory compliance
- **Core Services**: EKS, ECR, Secrets Manager, RDS PostgreSQL
- **Compliance**: Extensive compliance certifications for financial services
- **Hybrid Integration**: Seamless integration with on-premises systems when required

8.2.1.2 Core Services Requirements

Google Cloud Platform Services:

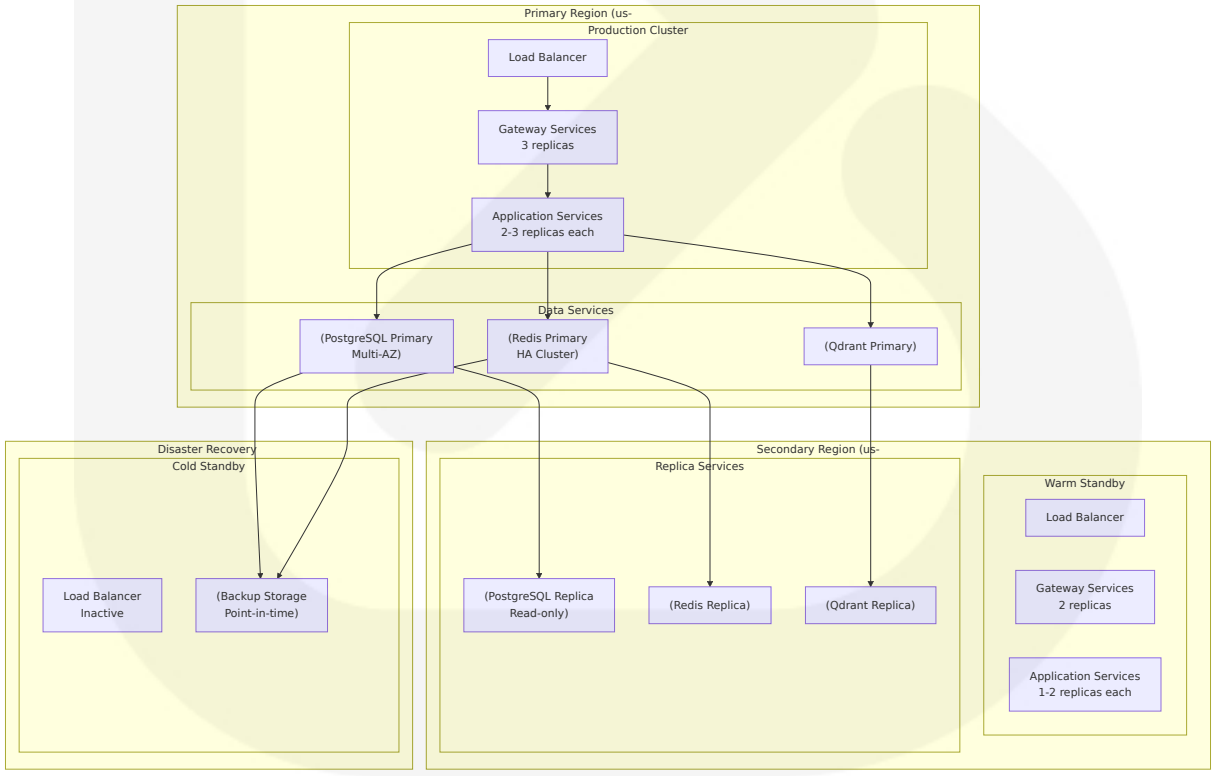
Service	Version/Configuration	Purpose	Scaling Configuration
Cloud Run	2nd generation	Serverless container hosting	0-10 instances, 100 concurrent requests
Artifact Registry	Standard tier	Container image storage	Multi-region replication
Secret Manager	Standard	Credential management	Automatic rotation enabled
Cloud SQL	PostgreSQL 14	Primary database	Read replicas, automated backups

Amazon Web Services Configuration:

Service	Configuration	Purpose	High Availability
EKS	Version 1.24+	Kubernetes orchestration	Multi-AZ deployment
ECR	Private repositories	Container registry	Cross-region replication
RDS PostgreSQL	14.x with encryption	Database service	Multi-AZ with read replicas
Secrets Manager	Automatic rotation	Credential storage	Cross-region backup

8.2.2 High Availability Design

8.2.2.1 Multi-Region Architecture



8.2.2.2 Auto-Scaling Configuration

Container Auto-Scaling (Cloud Run):

```
# Cloud Run scaling configuration
spec:
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/minScale: "1"
        autoscaling.knative.dev/maxScale: "10"
        autoscaling.knative.dev/targetConcurrencyUtilization: "70"
    spec:
      containerConcurrency: 100
      timeoutSeconds: 300
```

Kubernetes Auto-Scaling (EKS):

- **Horizontal Pod Autoscaler**: Scale based on CPU/memory utilization
- **Vertical Pod Autoscaler**: Automatic resource request optimization
- **Cluster Autoscaler**: Node scaling based on pod resource requirements
- **Custom Metrics**: Application-specific scaling based on request queue depth

8.2.3 Cost Optimization Strategy

8.2.3.1 Resource Optimization

Serverless Cost Management:

- **Cloud Run**: Pay-per-use pricing with automatic scale-to-zero capabilities
- **Request-Based Scaling**: Minimize idle costs through aggressive scale-down policies
- **Resource Right-Sizing**: Continuous monitoring and optimization of CPU/memory allocation

- **Regional Optimization**: Cost-aware region selection for non-critical workloads

Reserved Capacity Planning:

Service Category	Reservation Strategy	Expected Savings	Commitment Period
Database Services	Reserved instances for base load	30-50% cost reduction	1-3 years
Container Compute	Committed use discounts	20-30% savings	1 year
Storage Services	Lifecycle policies and archival	40-60% for cold storage	Ongoing
Network Services	CDN optimization	25-35% bandwidth reduction	Ongoing

8.2.3.2 Cost Monitoring Implementation

Real-Time Cost Tracking:

- **Service-Level Cost Attribution**: Detailed billing breakdown by microservice
- **Environment Cost Separation**: Clear delineation between dev, staging, and production costs
- **Alert Thresholds**: Automated alerts for unexpected cost spikes or budget overruns
- **Optimization Recommendations**: Automated suggestions for resource optimization

8.3 CONTAINERIZATION

8.3.1 Container Platform Selection

8.3.1.1 Docker Implementation Strategy

Container Platform Justification: Docker provides the foundational containerization technology with proven stability, extensive ecosystem support, and seamless integration with both Kubernetes and serverless platforms. The platform leverages Docker's multi-stage build capabilities for optimized production images.

Base Image Strategy:

```
# Multi-stage build example from Dockerfile
FROM node:20-alpine AS base
RUN apk add --no-cache libc6-compat && \
    addgroup --system --gid 1001 nodejs && \
    adduser --system --uid 1001 nextjs

FROM base AS deps
WORKDIR /app
COPY package.json pnpm-lock.yaml* ./
RUN corepack enable pnpm && pnpm install --frozen-lockfile

FROM base AS builder
WORKDIR /app
COPY --from=deps /app/node_modules ./node_modules
COPY . .
RUN pnpm run build

FROM base AS runner
WORKDIR /app
COPY --from=builder --chown=nextjs:nodejs /app/.next/standalone ./
USER nextjs
EXPOSE 3000
CMD ["node", "server.js"]
```

8.3.1.2 Image Versioning Approach

Semantic Versioning Strategy:

- **Release Tags**: Production deployments use semantic versioning (v1.0.0, v1.1.0)

- **Branch-Based Tags**: Staging deployments use branch names (main, develop)
- **Commit SHA Tags**: Development builds use short commit hashes for traceability
- **Latest Tags**: Convenience tags for latest stable and development images

Image Registry Distribution:

Registry	Purpose	Retention Policy	Access Control
GitHub Container Registry	Primary development and CI/CD	90 days for untagged	Repository-based access
Google Artifact Registry	Production Cloud Run deployments	1 year for tagged releases	IAM-based service accounts
AWS Elastic Container Registry	EKS production deployments	1 year for production images	Role-based access control

8.3.2 Build Optimization Techniques

8.3.2.1 Multi-Stage Build Architecture

Optimized Build Pipeline:

1. **Base Stage**: Common OS and runtime setup with security hardening
2. **Dependencies Stage**: Package installation with dependency caching
3. **Build Stage**: Application compilation and optimization
4. **Runtime Stage**: Minimal production image with only necessary artifacts

Build Optimization Results:

- **Image Size Reduction**: 60-70% smaller than single-stage builds

- **Build Time Optimization**: Layer caching reduces rebuild time by 50-80%
- **Security Surface**: Minimal attack surface with only production dependencies
- **Startup Performance**: Faster container startup with optimized image layers

8.3.2.2 Layer Caching Strategy

Docker Layer Optimization:

```
# Optimized layer ordering for maximum cache utilization
COPY package.json pnpm-lock.yaml* ./  # Cache dependencies separately
RUN pnpm install --frozen-lockfile     # Heavy operation cached when poss
COPY . .                               # Application code changes freque
RUN pnpm run build                     # Build step leverages previous ca
```

Registry Layer Sharing: Multi-service deployments leverage shared base layers, reducing storage requirements and deployment time across the microservices architecture.

8.3.3 Security Scanning Requirements

8.3.3.1 Vulnerability Scanning Integration

Trivy Security Scanner: Integrated into CI/CD pipeline for comprehensive vulnerability detection:

- **OS Package Scanning**: Detection of known vulnerabilities in Alpine Linux packages
- **Application Dependency Scanning**: Node.js and Python package vulnerability assessment
- **Configuration Scanning**: Docker and Kubernetes security best practices validation
- **License Compliance**: Open source license compatibility verification

Scanning Automation:

```
# GitHub Actions security scanning workflow
- name: Run Trivy vulnerability scanner
  uses: aquasecurity/trivy-action@master
  with:
    image-ref: ${{ env.IMAGE_NAME }}:${{ github.sha }}
    format: 'sarif'
    output: 'trivy-results.sarif'
    severity: 'CRITICAL,HIGH'
```

8.3.3.2 Container Hardening

Security Hardening Measures:

- **Non-Root Execution**: All containers run as non-privileged user (UID 1001)
- **Read-Only Root Filesystem**: Container filesystem mounted as read-only where possible
- **Minimal Package Installation**: Only essential packages included in production images
- **Security Updates**: Automated base image updates for security patches

8.4 ORCHESTRATION

8.4.1 Orchestration Platform Selection

8.4.1.1 Dual-Platform Strategy

Google Cloud Run (Primary):

- **Serverless Container Platform**: Eliminates infrastructure management overhead

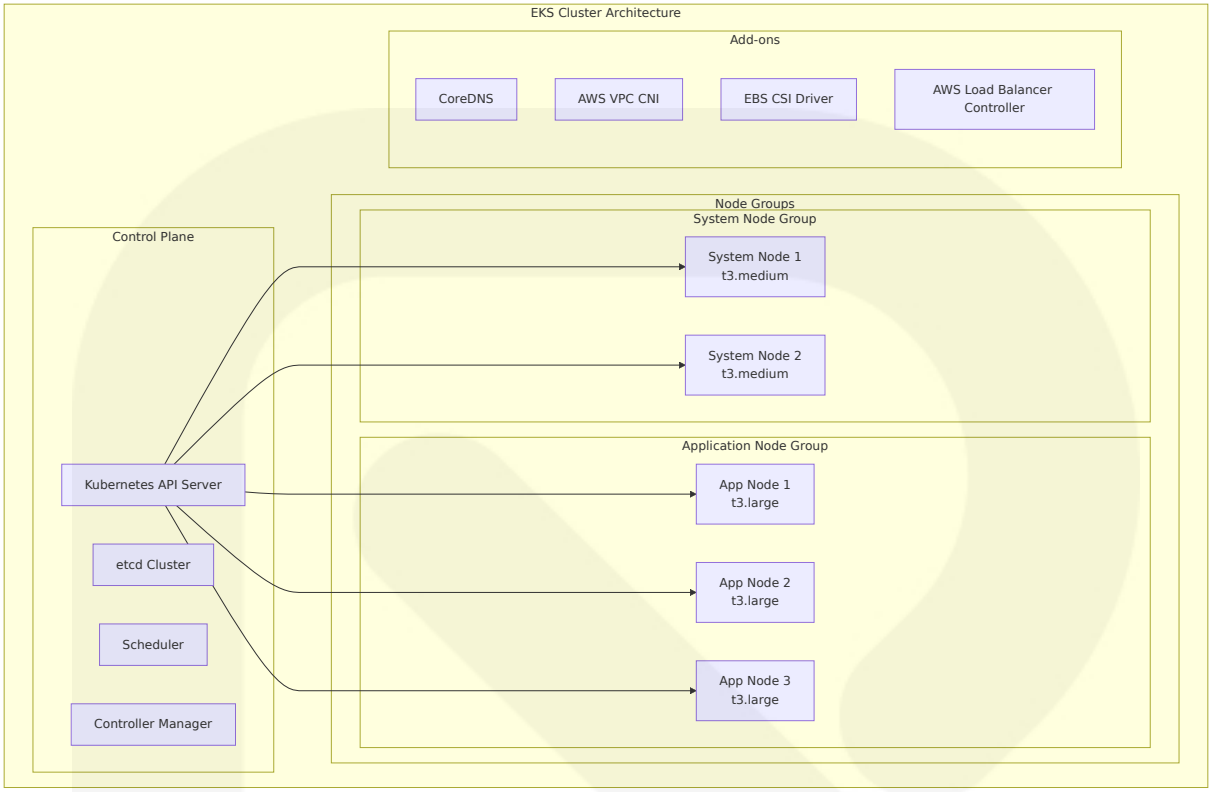
- **Automatic Scaling**: Request-driven scaling from zero to thousands of instances
- **Cost Efficiency**: Pay-per-request pricing model with sub-second billing
- **Managed Infrastructure**: Fully managed with built-in load balancing and SSL

Amazon EKS (Secondary):

- **Complex Orchestration Requirements**: Full Kubernetes control for advanced scenarios
- **Enterprise Integration**: Enhanced integration capabilities with existing enterprise systems
- **Compliance Requirements**: Additional compliance certifications for regulated environments
- **Custom Resource Management**: Advanced resource allocation and scheduling policies

8.4.1.2 Cluster Architecture

EKS Cluster Configuration:



8.4.2 Service Deployment Strategy

8.4.2.1 Helm Chart Management

Helm Chart Structure:

```
infra/helm/veria/
â”œâ”€â”€ Chart.yaml              # Chart metadata and dependencies
â”œâ”€â”€ values.yaml             # Default configuration values
â”œâ”€â”€ values-staging.yaml     # Staging environment overrides
â”œâ”€â”€ values-production.yaml # Production environment overrides
â””â”€â”€ templates/
    â”œâ”€â”€ gateway/           # Gateway service manifests
    â”œâ”€â”€ identity/          # Identity service manifests
    â”œâ”€â”€ compliance/        # Compliance service manifests
    â”œâ”€â”€ configmaps/       # Configuration management
    â””â”€â”€ secrets/          # Secret templates
```

Deployment Configuration:

Service	Staging Replicas	Production Replicas	Resource Limits
Gateway Service	1	3	1 CPU, 1Gi memory
Identity Service	1	2	0.5 CPU, 512Mi memory
Compliance Service	1	2	2 CPU, 2Gi memory
KYC Provider	1	2	1 CPU, 1Gi memory

8.4.2.2 Rolling Update Strategy

Kubernetes Deployment Configuration:

```
spec:
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 25%
      maxSurge: 25%
  template:
    spec:
      containers:
      - name: service
        livenessProbe:
          httpGet:
            path: /health
            port: 3000
          initialDelaySeconds: 30
          periodSeconds: 10
        readinessProbe:
          httpGet:
            path: /health
            port: 3000
          initialDelaySeconds: 5
          periodSeconds: 5
```

8.4.3 Auto-Scaling Configuration

8.4.3.1 Horizontal Pod Autoscaler

HPA Configuration:

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: gateway-service-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: gateway-service
  minReplicas: 2
  maxReplicas: 50
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 70
  - type: Resource
    resource:
      name: memory
      target:
        type: Utilization
        averageUtilization: 80
```

8.4.3.2 Resource Allocation Policies

Quality of Service Classes:

Service Category	QoS Class	CPU Request/Limit	Memory Request/Limit	Priority Class
Gateway Service	Guaranteed	500m/1000m	512Mi/1Gi	high-priority

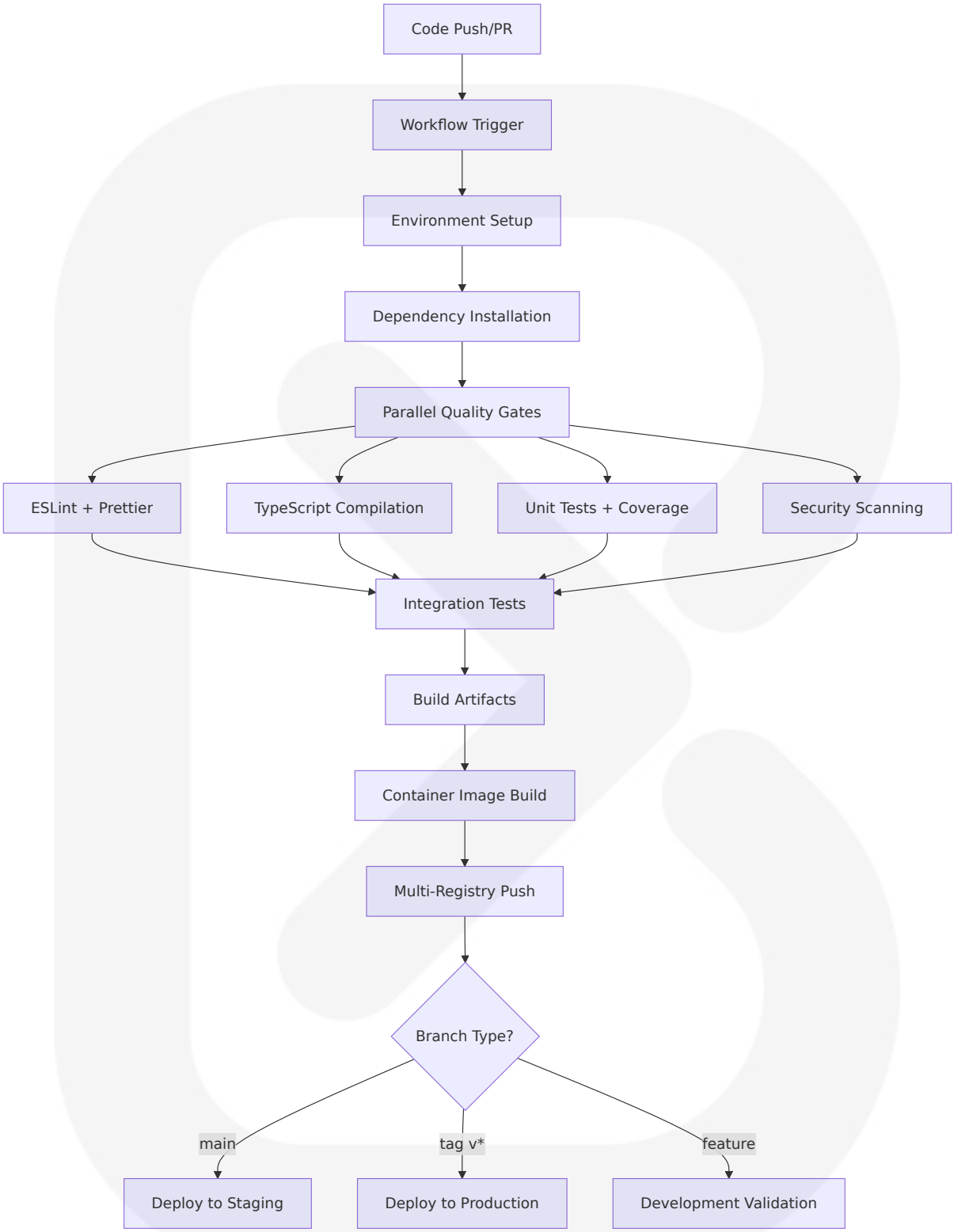
Service Category	QoS Class	CPU Request/Limit	Memory Request/Limit	Priority Class
Core Services	Guaranteed	250m/500m	256Mi/512Mi	medium-priority
Background Jobs	Burstable	100m/500m	128Mi/256Mi	low-priority
Development	BestEffort	No limits	No limits	development

8.5 CI/CD PIPELINE

8.5.1 Build Pipeline

8.5.1.1 GitHub Actions Workflow Architecture

Continuous Integration Pipeline:



Build Environment Requirements:

- **Node.js Runtime**: Version 20.x with pnpm package manager

- **Docker Engine**: Version 20.10+ for container builds
- **Build Resources**: 4 CPU cores, 8GB RAM for parallel builds
- **Storage**: 20GB for dependencies caching and artifacts

8.5.1.2 Source Control Integration

GitHub Integration Configuration:

- **Trigger Events**: Push to main/develop branches, pull request creation/updates
- **Branch Protection**: Required status checks for all quality gates
- **Merge Requirements**: All tests pass, code review approval, up-to-date branch
- **Automatic Cleanup**: Temporary deployments removed after branch deletion

Monorepo Build Optimization:

```
# GitHub Actions matrix strategy for parallel builds
strategy:
  matrix:
    service: [gateway, identity, compliance, kyc-provider, blockchain]
    environment: [staging, production]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
    - name: Build service
      run: pnpm --filter @veria/${{ matrix.service }} build
```

8.5.1.3 Dependency Management

pnpm Workspace Management:

- **Shared Dependencies**: Common packages installed at root level
- **Service Dependencies**: Service-specific packages managed independently

- **Build Dependencies**: Development tools shared across all services
- **Security Updates**: Automated dependency updates via Renovate Bot

Package Resolution Strategy:

- **Lockfile Management**: Single pnpm-lock.yaml for deterministic builds
- **Peer Dependencies**: Resolved at workspace root for consistency
- **Optional Dependencies**: Excluded from production builds
- **Audit Integration**: Security vulnerability checking in CI pipeline

8.5.1.4 Artifact Generation and Storage

Build Artifacts:

Artifact Type	Storage Location	Retention Policy	Purpose
Container Images	Multi-registry (GHCR, GAR, ECR)	90 days untagged, 1 year tagged	Deployment artifacts
npm Packages	GitHub Packages	6 months	Internal package distribution
Test Reports	GitHub Actions artifacts	30 days	Quality assurance records
Security Scans	SARIF format in repo	Permanent	Compliance documentation

8.5.1.5 Quality Gates

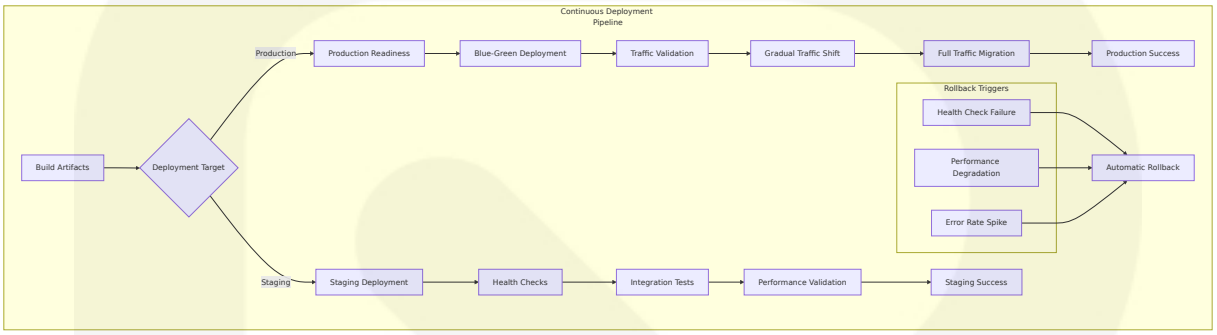
Automated Quality Validation:

1. **Code Style**: ESLint and Prettier for consistent formatting
2. **Type Safety**: TypeScript strict mode compilation
3. **Unit Testing**: Vitest with 80%+ code coverage requirement
4. **Integration Testing**: Containerized service testing with Docker Compose
5. **Security Scanning**: Trivy and npm audit for vulnerability detection
6. **Performance Testing**: k6 load testing for API endpoints

8.5.2 Deployment Pipeline

8.5.2.1 Deployment Strategy

Multi-Environment Deployment Architecture:



Blue-Green Deployment (Cloud Run):

```
# Cloud Run traffic allocation configuration
apiVersion: serving.knative.dev/v1
kind: Service
spec:
  traffic:
  - latestRevision: false
    revisionName: veria-gateway-00042-blue
    percent: 50
  - latestRevision: true
    revisionName: veria-gateway-00043-green
    percent: 50
```

8.5.2.2 Environment Promotion Workflow

Automated Promotion Gates:

Gate	Staging Requirements	Production Requirements	Validation Time
Health Checks	All services responding	All services healthy + 5min stability	2 minutes
Performance	<500ms p95 response time	<200ms p95 response time	10 minutes

Gate	Staging Requirements	Production Requirements	Validation Time
Integration	All APIs functional	External integrations validated	15 minutes
Security	No high/critical vulnerabilities	Security scan passed	5 minutes

8.5.2.3 Rollback Procedures

Automatic Rollback Triggers:

- **Health Check Failures**: 3 consecutive health check failures trigger rollback
- **Error Rate Thresholds**: >5% error rate for 2 minutes initiates rollback
- **Performance Degradation**: p95 latency >1000ms for 5 minutes triggers rollback
- **External Dependency Failures**: Cascading service failures activate rollback

Rollback Execution:

```
# Cloud Run rollback command
gcloud run services update-traffic veria-gateway \
  --to-revisions=veria-gateway-00042=100 \
  --region=us-central1
```

8.5.2.4 Post-Deployment Validation

Validation Test Suite:

1. **Smoke Tests**: Critical path functionality verification
2. **API Contract Tests**: Service interface validation
3. **Security Tests**: Authentication and authorization verification
4. **Performance Tests**: Response time and throughput validation

5. **Integration Tests**: External service connectivity validation

8.5.3 Release Management Process

8.5.3.1 Release Versioning

Semantic Versioning Strategy:

- **Major Versions**: Breaking API changes or significant architecture updates
- **Minor Versions**: New features and backward-compatible enhancements
- **Patch Versions**: Bug fixes and security updates
- **Pre-release Tags**: Beta and release candidate versions for testing

Release Branch Management:

- **Feature Branches**: Individual feature development
- **Develop Branch**: Integration branch for ongoing development
- **Main Branch**: Stable branch for staging deployments
- **Release Tags**: Production deployment triggers

8.5.3.2 Release Coordination

Multi-Service Release Coordination:

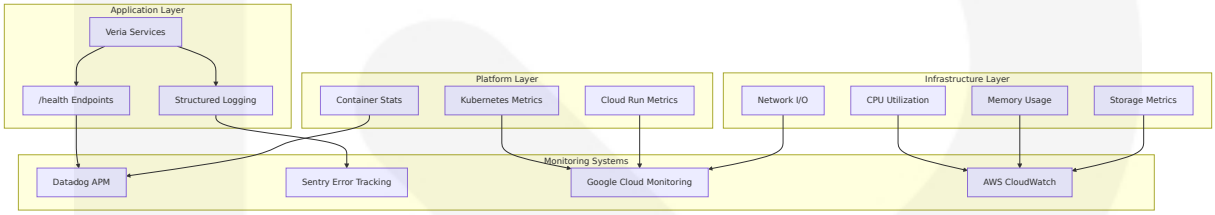
```
# Release coordination workflow
name: Coordinated Release
on:
  workflow_dispatch:
    inputs:
      release_version:
        description: 'Release version (e.g., v1.2.0)'
        required: true
      services:
        description: 'Comma-separated service names'
        default: 'gateway,identity,compliance'
```

8.6 INFRASTRUCTURE MONITORING

8.6.1 Resource Monitoring Approach

8.6.1.1 Observability Stack

Multi-Layer Monitoring Architecture:



8.6.1.2 Health Check Implementation

Comprehensive Health Monitoring:

- **Service Health**: HTTP endpoints returning service status and dependencies
- **Database Connectivity**: Connection pool status and query performance
- **Cache Availability**: Redis connectivity and memory utilization
- **External Dependencies**: KYC provider and blockchain network status

Health Check Configuration:

```
// Health check endpoint implementation
app.get('/health', async (request, reply) => {
  const checks = await Promise.allSettled([
    checkDatabase(),
    checkRedis(),
    checkExternalServices()
  ]);

  const healthy = checks.every(check => check.status === 'fulfilled');
  return reply.code(healthy ? 200 : 503).send({
```

```
      status: healthy ? 'healthy' : 'unhealthy',
      timestamp: new Date().toISOString(),
      checks: checks
    });
  });
```

8.6.2 Performance Metrics Collection

8.6.2.1 Application Performance Monitoring

Key Performance Indicators:

Metric Category	Specific Metrics	Target Values	Alert Thresholds
Response Time	p50, p95, p99 latency	<200ms p95	>500ms p95
Throughput	Requests per second	10,000+ TPS	<1,000 TPS
Error Rate	HTTP 4xx/5xx percentage	<1% error rate	>5% error rate
Availability	Service uptime	99.99% SLA	<99.9% uptime

Custom Metrics Collection:

- **Business Metrics**: Compliance checks per minute, KYC processing time
- **Security Metrics**: Authentication success/failure rates, blocked requests
- **Integration Metrics**: External API response times and success rates
- **Database Metrics**: Query performance, connection pool utilization

8.6.2.2 Infrastructure Performance Tracking

Resource Utilization Monitoring:

```
# Prometheus monitoring configuration
global:
  scrape_interval: 15s
  evaluation_interval: 15s

scrape_configs:
- job_name: 'veria-services'
  kubernetes_sd_configs:
  - role: pod
  relabel_configs:
  - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_scrape
    action: keep
    regex: true
```

8.6.3 Cost Monitoring and Optimization

8.6.3.1 Cost Tracking Implementation

Service-Level Cost Attribution:

- **Resource Tagging**: Consistent tagging strategy across all cloud resources
- **Service Mapping**: Direct correlation between services and infrastructure costs
- **Environment Segregation**: Clear cost separation between dev, staging, and production
- **Feature Cost Analysis**: Cost attribution to specific compliance features

Cost Optimization Alerts:

Alert Type	Threshold	Action	Notification
Daily Spend	>120% of budget	Scale down non-production	Email + Slack
Idle Resources	>1 hour unused	Auto-shutdown	Dashboard notification

Alert Type	Threshold	Action	Notification
Storage Growth	>150% of expected	Archive old data	Weekly report
Network Costs	>200% baseline	Review data transfer	Engineering team alert

8.6.3.2 Resource Optimization

Automated Cost Optimization:

- **Right-Sizing**: Continuous monitoring and adjustment of resource allocations
- **Reserved Capacity**: Analysis and recommendation for reserved instance purchases
- **Spot Instances**: Non-critical workloads moved to spot instances where appropriate
- **Data Lifecycle**: Automated archival and deletion of old audit logs and reports

8.6.4 Security Monitoring

8.6.4.1 Security Event Monitoring

Security Monitoring Implementation:

- **Authentication Monitoring**: Failed login attempts and suspicious patterns
- **Authorization Tracking**: Privilege escalation attempts and access violations
- **Network Security**: Intrusion detection and DDoS protection
- **Data Access Monitoring**: Unusual data access patterns and bulk downloads

Security Alerting Configuration:

```
# Security alert rules
groups:
- name: security.rules
  rules:
  - alert: SuspiciousLoginActivity
    expr: failed_auth_attempts > 10
    for: 5m
    labels:
      severity: critical
    annotations:
      summary: "Suspicious login activity detected"

  - alert: UnauthorizedDataAccess
    expr: unauthorized_data_requests > 5
    for: 2m
    labels:
      severity: high
```

8.6.4.2 Compliance Auditing

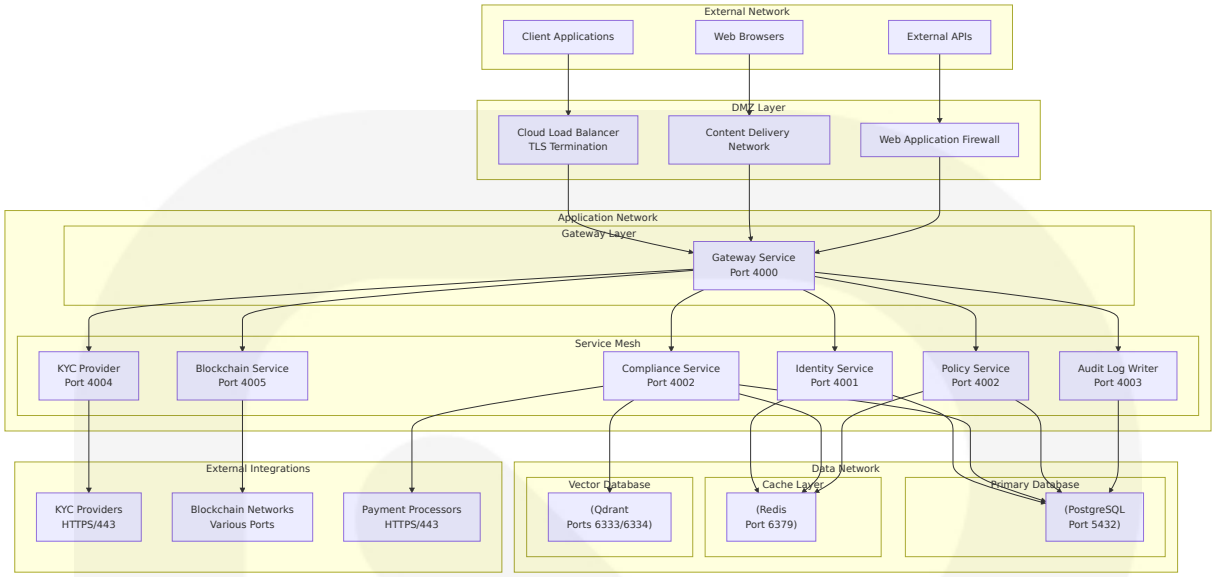
Audit Trail Monitoring:

- **Regulatory Compliance**: Automated compliance checking against SOX, GDPR requirements
- **Data Retention**: Monitoring and enforcement of data retention policies
- **Access Auditing**: Comprehensive logging of all data access and modifications
- **Change Tracking**: Configuration and code change monitoring with approval workflows

8.7 NETWORK ARCHITECTURE

8.7.1 Network Topology

8.7.1.1 Service Mesh Architecture



8.7.1.2 Security Zones

Network Segmentation Strategy:

Security Zone	Purpose	Access Control	Monitoring Level
DMZ	External-facing services	Public internet access	High
Application Tier	Business logic services	Internal service mesh only	Medium
Data Tier	Database and storage	Application tier only	High
Management	Administrative access	VPN and bastion hosts	Critical

8.7.2 Traffic Flow Patterns

8.7.2.1 Request Routing

Gateway-Centric Routing:

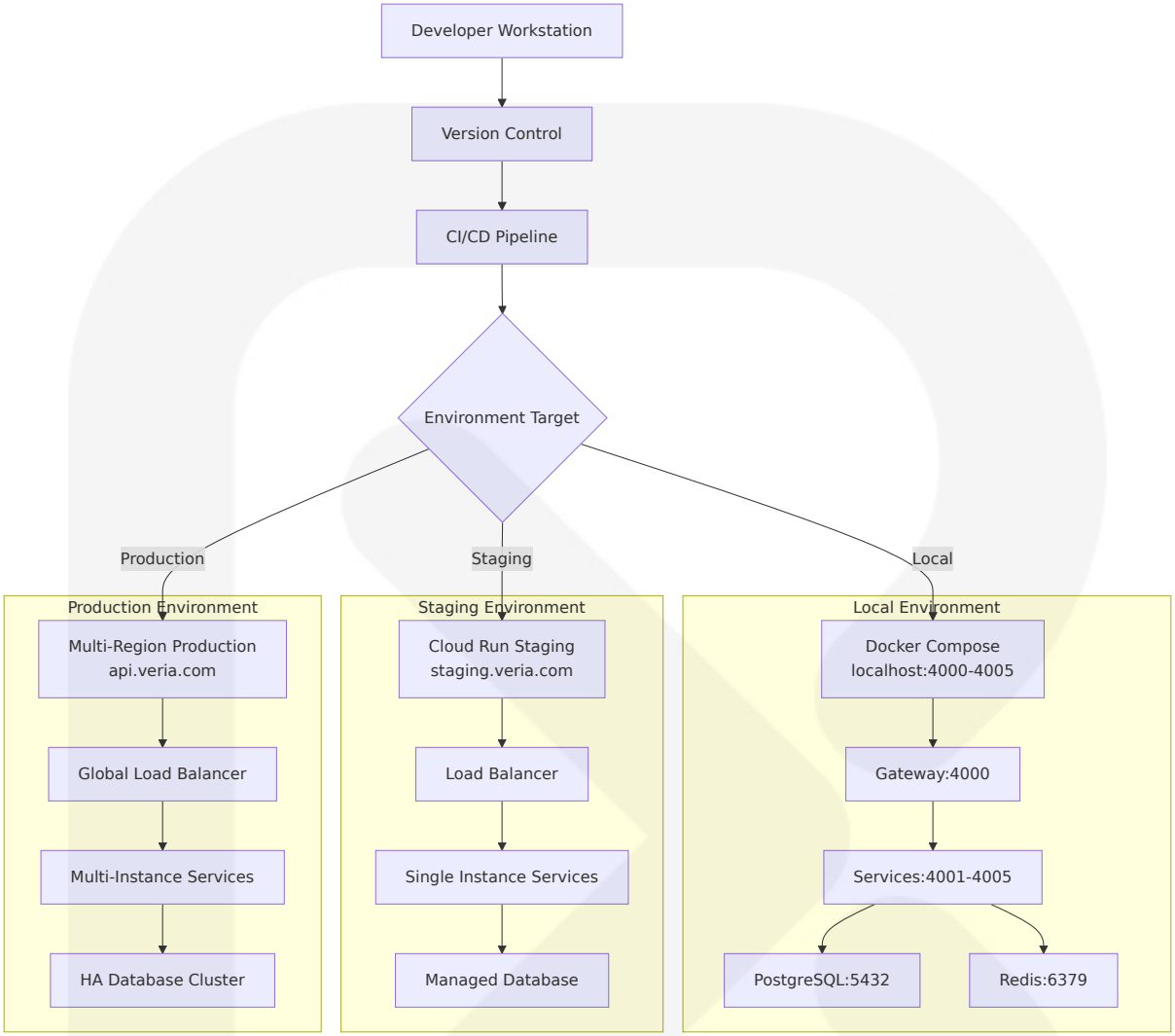
All external requests flow through the Gateway service which implements:

- **Authentication Validation**: JWT token verification and session validation
- **Rate Limiting**: Request throttling based on user and IP address
- **Load Balancing**: Distribution across service instances
- **Request Tracing**: Correlation ID injection for distributed tracing

Service Discovery:

```
// Gateway service routing configuration
const serviceRoutes = {
  '/api/auth': 'http://identity-service:4001',
  '/api/compliance': 'http://compliance-service:4002',
  '/api/policies': 'http://policy-service:4002',
  '/api/kyc': 'http://kyc-provider:4004',
  '/api/blockchain': 'http://blockchain-service:4005'
};
```

8.7.2.2 Environment Promotion Flow



8.8 INFRASTRUCTURE COST ESTIMATES

8.8.1 Environment Cost Breakdown

8.8.1.1 Monthly Infrastructure Costs

Development Environment:

- **Local Development**: $0 (Docker Compose on developer machines)

- **Shared Development Database**: $50-100/month (Cloud SQL small instance)
- **Development Container Registry**: $10-20/month (storage and bandwidth)

Staging Environment:

Component	Service	Monthly Cost	Justification
Compute	Cloud Run (1-2 instances)	$100-200	Auto-scaling serverless containers
Database	Cloud SQL PostgreSQL	$200-300	Small instance with backups
Cache	Redis (Memorystore)	$50-100	Basic Redis instance
Storage	Persistent disks	$50-100	Database and log storage
Network	Load balancing + egress	$50-100	Traffic and external API calls
Monitoring	Stackdriver/Datadog	$100-150	APM and logging
Total Staging		**$550-1,050**	

Production Environment:

Component	Service	Monthly Cost	Justification
Compute	Cloud Run (10+ instances)	$1,000-2,000	Multi-region auto-scaling
Database	Cloud SQL HA PostgreSQL	$800-1,500	High availability with read replicas
Cache	Redis Cluster	$300-500	High availability Redis

Component	Service	Monthly Cost	Justification
Storage	SSD persistent disks	$200-400	Fast storage for databases
Network	Global load balancing	$200-500	Multi-region traffic distribution
Monitoring	Full observability stack	$500-800	Comprehensive monitoring and APM
Security	WAF, DDoS protection	$100-200	Enhanced security services
Backup	Cross-region backups	$100-200	Disaster recovery storage
Total Production		**$3,200-6,100**	

8.8.1.2 Scaling Cost Projections

Cost Scaling by User Volume:

User Tier	Monthly Active Users	Infrastructure Cost	Cost Per User
Startup	0-1,000	$4,000-7,000	$4.00-7.00
Growth	1,000-10,000	$8,000-15,000	$1.50-8.00
Enterprise	10,000-100,000	$20,000-50,000	$0.20-5.00
Scale	100,000+	$50,000+	$0.50+

8.8.2 Cost Optimization Opportunities

8.8.2.1 Reserved Capacity Planning

1-Year Reserved Instance Savings:

- **Database Services**: 30-40% savings on consistent base load

- **Compute Resources**: 20-30% savings on minimum instance requirements
- **Storage**: 15-25% savings on data retention requirements
- **Network**: 10-20% savings on committed bandwidth usage

3-Year Commitment Benefits:

- **Additional Savings**: 15-25% beyond 1-year pricing
- **Budget Predictability**: Fixed costs for long-term planning
- **Capacity Guarantees**: Reserved capacity during high-demand periods

8.8.2.2 Auto-Scaling Optimization

Dynamic Resource Management:

- **Scale-to-Zero**: Development and staging environments auto-stop after hours
- **Load-Based Scaling**: Production auto-scaling based on actual demand
- **Scheduled Scaling**: Predictable traffic patterns with pre-scaling
- **Geographic Optimization**: Traffic routing to lowest-cost regions

References

Infrastructure Files Examined

- `Dockerfile` - Multi-stage build configuration for Node.js services with security hardening
- `docker-compose.yml` - Complete development environment orchestration with service dependencies
- `cloudrun.yaml` - Google Cloud Run deployment manifest with auto-scaling configuration
- `.github/workflows/ci.yml` - Continuous integration pipeline with quality gates

- `.github/workflows/cd.yml` - Continuous deployment to Kubernetes/EKS clusters
- `.github/workflows/deploy-backend.yml` - Cloud Run deployment workflow with health checks
- `.env.example` - Environment variable template with service configurations
- `BLITZY_SETUP.md` - Local setup and deployment instructions
- `package.json` - Monorepo scripts and dependency management
- `pnpm-workspace.yaml` - Workspace configuration for monorepo builds

Infrastructure Directories Analyzed

- `/` (repository root) - Infrastructure configuration files and deployment manifests
- `infra/` - Infrastructure as Code configurations and deployment templates
- `infra/database/` - Database schemas, migrations, and initialization scripts
- `infra/docker/` - Additional Docker configurations and multi-stage build optimizations
- `infra/helm/` - Kubernetes Helm charts for EKS deployment
- `.github/` - GitHub configuration and workflow definitions
- `.github/workflows/` - CI/CD pipeline definitions and deployment automation
- `ops/` - Operational playbooks, runbooks, and maintenance procedures
- `scripts/` - Infrastructure setup scripts and automation tools

Technical Specification Sections Referenced

- `5.1 HIGH-LEVEL ARCHITECTURE` - System architecture context and microservices design principles
- `6.1 CORE SERVICES ARCHITECTURE` - Service mesh patterns, auto-scaling, and resilience implementation

- `6.4 SECURITY ARCHITECTURE` - Security zones, encryption standards, and compliance requirements
- `2.5 NON-FUNCTIONAL REQUIREMENTS` - Performance targets, scalability requirements, and SLA definitions
- `3.6 DEVELOPMENT & DEPLOYMENT` - Container strategies, build systems, and deployment platforms

APPENDICES

9. APPENDICES

9.1 ADDITIONAL TECHNICAL INFORMATION

9.1.1 Development Environment Setup Requirements

9.1.1.1 Runtime Environment Dependencies

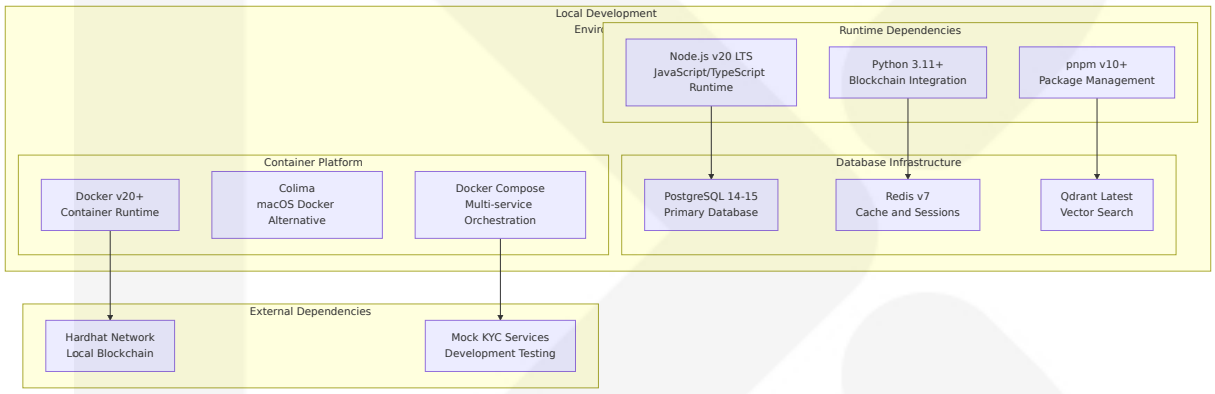
The Veria platform requires specific runtime versions and tools for consistent development environments across team members and CI/CD pipelines.

Component	Specification	Installation Method	Purpose	Validation Command
Node.js Runtime	v20 LTS	nvm or official installer	Primary runtime for TypeScript services	`node --version`
Python Runtime	v3.11+	pyenv recommended	Blockchain and compliance packages	`python --version`

Component	Specification	Installation Method	Purpose	Validation Command
pnpm Package Manager	v10+	`corepack enable`	Workspace-aware monorepo management	`pnpm --version`
PostgreSQL Database	v14-15	Docker or native install	Primary data storage	`psql --version`

9.1.1.2 Development Tools and Extensions

Required Development Infrastructure:



9.1.2 Service Port Allocation Matrix

9.1.2.1 Production Service Ports

The following port allocation ensures consistent service discovery and load balancing across deployment environments:

Service	Port	Environment Variable	Protocol	Purpose
Gateway	4000	GATEWAY_PORT	HTTP/HTTPS	API gateway and request routing

Service	Port	Environment Variable	Protocol	Purpose
Identity Service	4001	IDENTITY_SERVICE_PORT	HTTP/HTTPS	Authentication and authorization
Compliance Service	4002	COMPLIANCE_SERVICE_PORT	HTTP/HTTPS	Compliance rule evaluation
Audit Service	4003	AUDIT_SERVICE_PORT	HTTP/HTTPS	Audit log writing and retrieval
KYC Provider	4004	KYC_SERVICE_PORT	HTTP/HTTPS	KYC/KYB verification orchestration
Blockchain Service	4005	BLOCKCHAIN_SERVICE_PORT	HTTP/HTTPS	Smart contract interactions

9.1.2.2 Frontend and Database Ports

Component	Port	Environment Variable	Purpose	Access Level
Frontend Application	3000	-	Next.js main web application	Public
Compliance Dashboard	3010	-	React compliance management UI	Internal
PostgreSQL Database	5432	DATABASE_URL	Primary data connections	Internal
Redis Cache	6379	REDIS_URL	Session and cache management	Internal
Qdrant HTTP API	6333	QDRANT_HTTP_URL	Vector search HTTP interface	Internal
Qdrant gRPC API	6334	QDRANT_GRPC_URL	High-performance vector ope	Internal

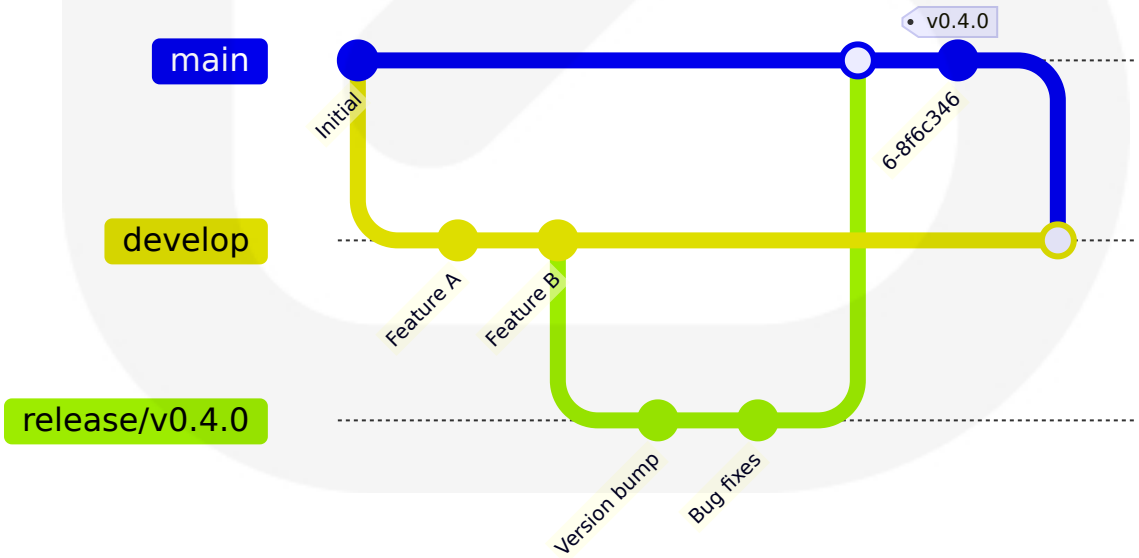
Component	Port	Environment Variable	Purpose	Access Level
			rations	

9.1.3 Repository Version Management

9.1.3.1 Versioning Strategy

Marker Type	Location	Current Value	Update Trigger	Purpose
Repository Version	VERSION.txt	v3	Major architecture changes	Overall system versioning
Package Version	package.json	0.4.0	Feature releases	Workspace version coordination
Database Schema	Alembic migrations	Auto-increment	Schema changes	Database version tracking
API Specification	OpenAPI spec	3.0.3	API contract updates	API version management

9.1.3.2 Release Branch Strategy



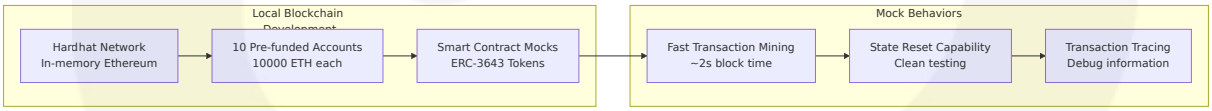
9.1.4 Mock Service Configuration Matrix

9.1.4.1 Development Mock Services

For development environments, the platform provides deterministic mock implementations that simulate external service behaviors without requiring API keys or network dependencies:

Mock Service	Configuration Key	Mock Behavior	Response Patterns
Chainalysis Mock	CHAINALYSIS_API_KEY=mock	Returns deterministic risk scores based on wallet address patterns	Low risk (0.1), Medium risk (0.5), High risk (0.9)
TRM Labs Mock	TRM_API_KEY=mock	Simulates transaction monitoring with configurable alerts	Clean (no alerts), Suspicious (flagged), Blocked (high risk)
Jumio Mock	JUMIO_API_TOKEN=mock	Provides test identity verification with known outcomes	Pass (valid ID), Fail (invalid), Review (manual check)
Onfido Mock	ONFIDO_API_TOKEN=mock	Simulates document verification and liveness detection	Verified (authentic), Rejected (fraudulent), Inconclusive

9.1.4.2 Blockchain Mock Configuration

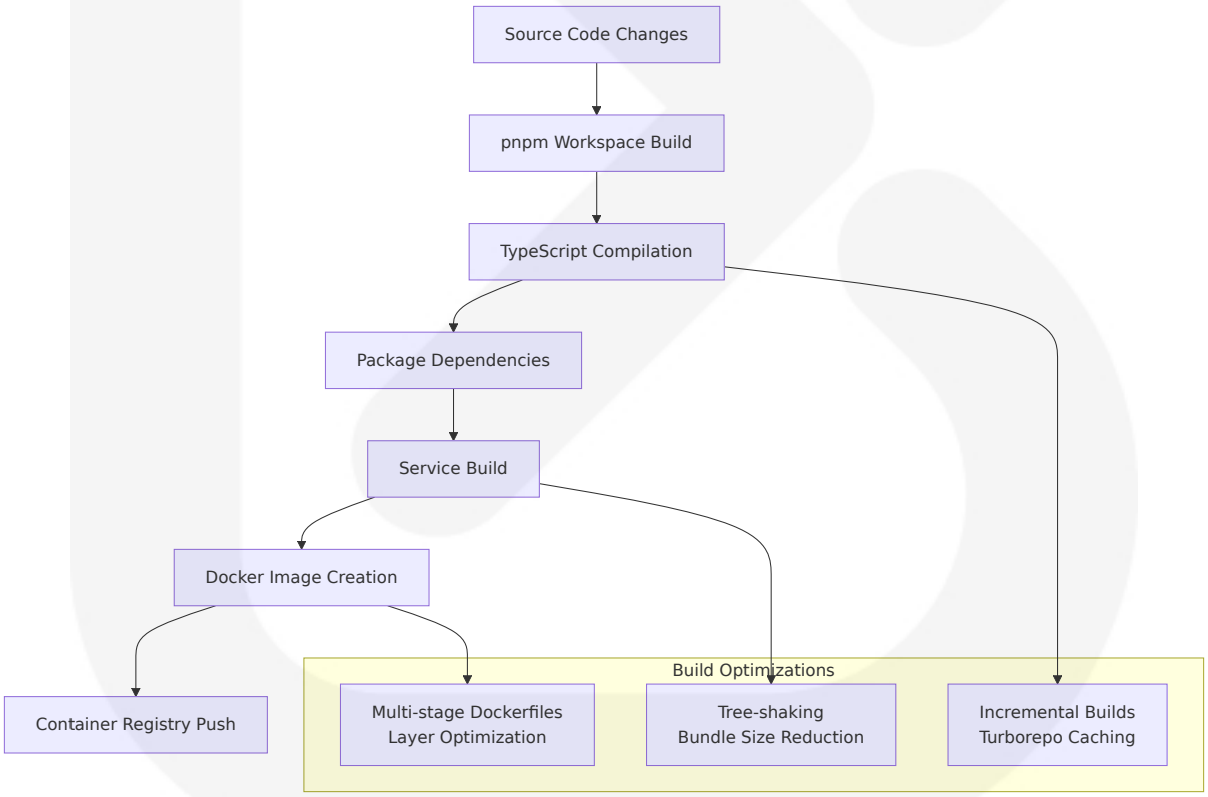


9.1.5 Build System Configuration

9.1.5.1 Compilation Targets by Language

Language/ Framework	Compilation Target	Module System	Output Directory	Source Maps
TypeScript Services	ES2020 (Node.js services)	ESM	dist/	Enabled in development
React Applications	ES2015+ (Browser compatibility)	ESM	build/ or dist/	Production optimized
Python Packages	Python 3.11 bytecode	Python modules	**pycache**/	Not applicable
Solidity Contracts	EVM bytecode (Ethereum)	-	artifacts/	Hardhat debugging

9.1.5.2 Build Pipeline Architecture

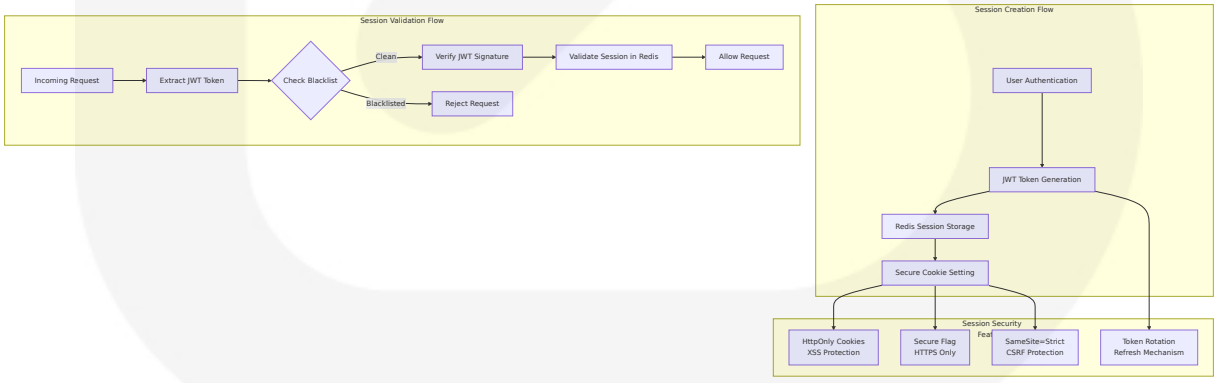


9.1.6 Session Management Architecture

9.1.6.1 Redis Session Storage Patterns

Session Type	Storage Key Pattern	TTL Duration	Refresh Strategy	Security Features
User Sessions	`session:<sessionId>`	7 days	Sliding expiration on activity	JWT hash storage, not plaintext
Refresh Tokens	`refresh:<refreshToken>`	7 days	Single-use with rotation	Secure random generation
JWT Blacklist	`blacklist:<jti>`	24 hours	Fixed expiration matching JWT	Distributed revocation
Rate Limit Windows	`rate:<ip>:<endpoint>`	60 seconds	Rolling window counters	IP-based throttling
Policy Cache	`policy:<ruleId>:<hash>`	300 seconds	Invalidation on rule changes	Compliance decision caching
KYC Results Cache	`kyc:<userId>:<provider>`	1 hour	Provider-specific TTL	Verification result optimization

9.1.6.2 Session Security Configuration

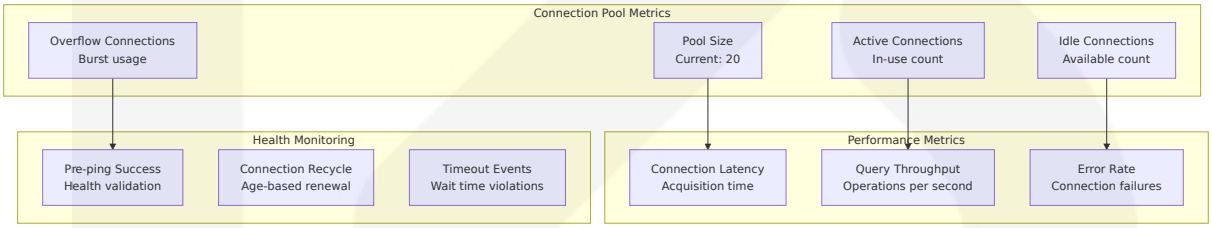


9.1.7 Database Connection Pool Optimization

9.1.7.1 SQLAlchemy Configuration Matrix

Setting	Value	Purpose	Impact	Monitoring Metric
pool_size	20	Base connection pool size	Memory usage baseline	Active connections
max_overflow	10	Additional connections under load	Burst capacity handling	Pool overflow events
pool_timeout	2000ms	Connection acquisition timeout	Request timeout prevention	Connection wait time
pool_recycle	3600s	Connection recycling interval	Stale connection prevention	Connection age
pool_pre_ping	true	Connection health validation	Failed query prevention	Health check success rate

9.1.7.2 Connection Pool Monitoring



9.1.8 Container Registry Configuration

9.1.8.1 Multi-Registry Strategy

Registry	Primary Use	Access Method	Repository Pattern	Backup Strategy
GitHub Container Registry	Development builds	GitHub Actions	ghcr.io/org/service:tag	Primary development

Registry	Primary Use	Access Method	Repository Pattern	Backup Strategy
Google Artifact Registry	Production GCP	Service account	region-docker.pkg.dev/project/repo/service	Primary production
Amazon ECR	Production AWS	IAM roles	account.dkr.ecr.region.amazonaws.com/service	Alternative production
Docker Hub	Public images	API tokens	dockerhub/org/service:tag	Emergency fallback

9.2 GLOSSARY

9.2.1 Financial and Regulatory Terms

Term	Definition
Accredited Investor	An individual or entity meeting specific financial criteria (minimum $1 million net worth or $200,000+ annual income) qualifying them to invest in private securities and unregistered offerings under SEC regulations
Anti-Money Laundering (AML)	Regulatory framework and processes designed to prevent criminals from disguising illegally obtained funds as legitimate income through financial transactions
Audit Trail	A chronological record of all system activities and transactions that provides documentary evidence for compliance reviews, regulatory audits, and forensic investigations
Compliance Decision	The automated or manual outcome of evaluating an entity, transaction, or activity against applicable regulatory rules, policies, and jurisdiction-specific requirements

Term	Definition
Custody Provider	A licensed financial institution (such as BNY Mellon) that holds, safeguards, and manages financial assets on behalf of clients while providing reporting and administration services
Know Your Business (KYB)	The process of verifying the identity, legitimacy, and beneficial ownership of business entities to comply with regulatory requirements and prevent fraudulent activities
Know Your Customer (KYC)	The mandatory process of verifying the identity of individual clients and assessing their risk profile to prevent fraud, money laundering, and terrorist financing
Qualified Purchaser	An investor meeting higher net worth requirements ($5 million+ for individuals, $25 million+ for entities) defined by the Investment Company Act of 1940, allowing investment in certain private funds
Real-World Asset (RWA)	Physical or traditional financial assets (such as real estate, commodities, bonds, or treasury bills) that are represented digitally on blockchain networks through tokenization
Securities and Exchange Commission (SEC)	The U.S. federal agency responsible for enforcing securities laws, regulating the securities markets, and protecting investors in financial markets
Special Purpose Vehicle (SPV)	A separate legal entity created for specific, limited business purposes, often used to isolate financial risk and hold assets for tokenization while providing legal and tax benefits
Suspicious Activity Report (SAR)	A document that financial institutions must file with authorities when they detect potentially illegal or suspicious financial activities that may indicate money laundering or fraud

9.2.2 Technical Architecture Terms

Term	Definition
API Gateway Pattern	An architectural pattern that provides a single entry point for multiple backend services, handling routing, authentication, rate limiting, and request/response transformation
Bearer Token	A type of access token used in authentication schemes where possession of the token grants access to resources, typically implemented using JWT (JSON Web Tokens)
Circuit Breaker	A design pattern that prevents cascading failures in distributed systems by detecting service failures and temporarily blocking requests to failing services until recovery
Dual-Write Pattern	A data persistence strategy where critical data is written to multiple storage systems simultaneously to ensure durability and enable audit trail requirements
Event Sourcing	An architectural pattern where all changes to application state are stored as a sequence of immutable events, enabling complete audit trails and state reconstruction
Fallback Provider	A secondary service provider automatically used when the primary provider is unavailable, fails, or exceeds response time thresholds, ensuring system resilience
Immutable Audit Log	A write-once, read-many audit record system where entries cannot be modified or deleted after creation, ensuring regulatory compliance and data integrity
Microservices Architecture	An architectural approach where applications are built as a collection of loosely coupled, independently deployable services that communicate over well-defined APIs
Multi-Signature (Multi-Sig)	A cryptographic security mechanism requiring multiple private key signatures to authorize blockchain transactions, providing enhanced security and governance
Rate Limiting	The practice of controlling the frequency of requests to an API or service to prevent abuse, ensure fair resource usage, and maintain system performance

Term	Definition
Service Mesh	An infrastructure layer that manages service-to-service communication in microservices architectures, providing features like load balancing, security, and observability
Vector Search	A similarity-based search technique using mathematical vectors and machine learning embeddings to find semantically related content in documents and data
Zero-Trust Architecture	A security model that requires verification for every user and device attempting to access resources, regardless of their location or network connection

9.2.3 Blockchain and Tokenization Terms

Term	Definition
ERC-3643 Token Standard	An Ethereum token standard specifically designed for security tokens, providing built-in compliance features, transfer restrictions, and regulatory controls
Gas Optimization	Techniques and practices used to reduce the computational cost (gas fees) of blockchain transactions by optimizing smart contract code and transaction batching
Hard Fork	A permanent change to blockchain protocol rules that creates two separate blockchain versions, requiring all network participants to upgrade to remain compatible
Identity Registry	A smart contract or centralized database that maintains verified identity information and compliance status for participants in a tokenized ecosystem
Modular Compliance	A flexible smart contract architecture that allows different compliance rules to be applied dynamically based on jurisdiction, asset type, or investor classification
Smart Contract	Self-executing contracts with terms directly written into code that automatically execute when predetermined conditions are met on a blockchain network

Term	Definition
Token Blacklisting	The process of invalidating or blocking specific tokens, addresses, or accounts from participating in transactions, typically for compliance or security reasons
Tokenization	The process of converting ownership rights in physical or financial assets into digital tokens that can be stored, transferred, and traded on blockchain networks
Transaction Batching	The practice of grouping multiple transactions together for processing as a single unit to improve efficiency and reduce costs on blockchain networks
Web3 Integration	The connection and interaction between traditional applications and decentralized blockchain networks, enabling hybrid centralized-decentralized architectures

9.2.4 Development and Operations Terms

Term	Definition
Blue-Green Deployment	A deployment strategy that maintains two identical production environments, allowing instant rollback and zero-downtime deployments by switching traffic between environments
Container Orchestration	The automated management, scaling, and networking of containerized applications using platforms like Kubernetes or Docker Swarm
Continuous Integration/Continuous Deployment (CI/CD)	Development practices that automate the integration of code changes and deployment to production environments through automated testing and deployment pipelines
Infrastructure as Code (IaC)	The practice of managing and provisioning computing infrastructure through machine-readable definition files rather than manual processes
Mean Time To Resolution (MTTR)	The average time required to fix a failed system or resolve an incident, measured from when the issue is first detected until it is fully resolved

Term	Definition
Observability	The ability to measure and understand the internal state of a system based on the data it generates, including metrics, logs, and distributed traces
Rolling Deployment	A deployment strategy that gradually replaces instances of the previous version with new versions, ensuring continuous service availability during updates
Service Level Agreement (SLA)	A commitment between a service provider and client that defines the expected level of service, including availability, performance, and response time metrics
Site Reliability Engineering (SRE)	A discipline that incorporates software engineering approaches to infrastructure and operations problems to create scalable and reliable software systems

9.3 ACRONYMS

9.3.1 Technical Acronyms

Acronym	Expanded Form
ACID	Atomicity, Consistency, Isolation, Durability
AES	Advanced Encryption Standard
API	Application Programming Interface
APM	Application Performance Monitoring
APY	Annual Percentage Yield
ASGI	Asynchronous Server Gateway Interface
AWS	Amazon Web Services
CI/CD	Continuous Integration/Continuous Deployment
CORS	Cross-Origin Resource Sharing

Acronym	Expanded Form
CPU	Central Processing Unit
CRUD	Create, Read, Update, Delete
DDL	Data Definition Language
DNS	Domain Name System
ECR	Elastic Container Registry
EKS	Elastic Kubernetes Service
ELK	Elasticsearch, Logstash, Kibana
ERC	Ethereum Request for Comments
ESM	ECMAScript Modules
EVM	Ethereum Virtual Machine
GAR	Google Artifact Registry
GCP	Google Cloud Platform
GHCR	GitHub Container Registry
GIN	Generalized Inverted Index
gRPC	Google Remote Procedure Call
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
I/O	Input/Output
IP	Internet Protocol
JSON	JavaScript Object Notation
JSONB	JSON Binary
JWT	JSON Web Token
LTS	Long Term Support
MFA	Multi-Factor Authentication
NPM	Node Package Manager

Acronym	Expanded Form
ORM	Object-Relational Mapping
OS	Operating System
PDF	Portable Document Format
REST	Representational State Transfer
RPC	Remote Procedure Call
RPO	Recovery Point Objective
RTO	Recovery Time Objective
S3	Simple Storage Service
SDK	Software Development Kit
SLA	Service Level Agreement
SLO	Service Level Objective
SQL	Structured Query Language
SRE	Site Reliability Engineering
SSH	Secure Shell
SSL	Secure Sockets Layer
TLS	Transport Layer Security
TTL	Time To Live
UI	User Interface
URL	Uniform Resource Locator
UUID	Universally Unique Identifier
UX	User Experience
VPC	Virtual Private Cloud
WAL	Write-Ahead Logging
YAML	Yet Another Markup Language

9.3.2 Financial and Compliance Acronyms

Acronym	Expanded Form
AML	Anti-Money Laundering
BSA	Bank Secrecy Act
CTR	Currency Transaction Report
FIDO2	Fast Identity Online 2
FINRA	Financial Industry Regulatory Authority
GDPR	General Data Protection Regulation
KYB	Know Your Business
KYC	Know Your Customer
NAV	Net Asset Value
OFAC	Office of Foreign Assets Control
PCI DSS	Payment Card Industry Data Security Standard
PII	Personally Identifiable Information
RBAC	Role-Based Access Control
REIT	Real Estate Investment Trust
RWA	Real-World Asset
SAR	Suspicious Activity Report
SEC	Securities and Exchange Commission
SOX	Sarbanes-Oxley Act
SPV	Special Purpose Vehicle
USD	United States Dollar

9.3.3 Business and Organizational Acronyms

Acronym	Expanded Form
B2B	Business to Business
CEO	Chief Executive Officer

Acronym	Expanded Form
CFO	Chief Financial Officer
CTO	Chief Technology Officer
KPI	Key Performance Indicator
MOU	Memorandum of Understanding
NDA	Non-Disclosure Agreement
P2P	Peer to Peer
QA	Quality Assurance
SMB	Small and Medium Business
SME	Subject Matter Expert

9.3.4 Blockchain and Cryptocurrency Acronyms

Acronym	Expanded Form
DeFi	Decentralized Finance
dApp	Decentralized Application
DAO	Decentralized Autonomous Organization
DLT	Distributed Ledger Technology
NFT	Non-Fungible Token
TVL	Total Value Locked
Web3	Decentralized Web

9.3.5 Monitoring and Operations Acronyms

Acronym	Expanded Form
APM	Application Performance Monitoring
MTBF	Mean Time Between Failures

Acronym	Expanded Form
MTTD	Mean Time To Detection
MTTR	Mean Time To Resolution
MTBSI	Mean Time Between Service Incidents
QoS	Quality of Service
SLI	Service Level Indicator
TSDB	Time Series Database

References

Files Examined

- `.env.example` - Environment variable configuration template with comprehensive service and database settings
- `scripts/init-db.js` - Database initialization script with DDL provisioning and development seed data
- `contracts/hardhat.config.js` - Smart contract development configuration for local blockchain testing
- `VERSION.txt` - Repository version marker for tracking major system releases
- `package.json` - Root workspace configuration with version management and dependency coordination

Folders Explored

- `` (depth: 1) - Repository root with monorepo structure and configuration files including Docker Compose and environment templates
- `packages` (depth: 1) - Eight shared packages including auth-middleware, blockchain, database, sdk-ts, and compliance middleware
- `scripts` (depth: 1) - Developer operation utilities including database initialization and development setup scripts

- `infra` (depth: 1) - Infrastructure artifacts including Docker configurations, Helm charts, and database schemas
- `ops` (depth: 1) - Operational runbooks, playbooks, and monitoring configurations for production management
- `contracts` (depth: 1) - Hardhat-based Solidity smart contract workspace with ERC-3643 token implementations
- `docs` (depth: 1) - Comprehensive documentation including architecture specifications, roadmap, and sprint artifacts
- `services` (depth: 1) - Eleven TypeScript microservices implementing consistent patterns for the compliance platform
- `tests` (depth: 0) - Multi-runtime test assets including end-to-end testing, performance validation, and k6 load testing
- `apps` (depth: 0) - Frontend applications including the main Next.js application and React compliance dashboard
- `.github` (depth: 0) - CI/CD workflows and GitHub Actions configurations for automated testing and deployment

Technical Specification Sections Referenced

- **1.2 SYSTEM OVERVIEW** - System context, capabilities, and technical approach providing foundation for appendix content
- **2.1 FEATURE CATALOG** - Complete feature inventory with implementation status and technical dependencies
- **3.1 PROGRAMMING LANGUAGES** - Language selection and multi-language architecture justification
- **3.3 OPEN SOURCE DEPENDENCIES** - Core runtime dependencies and development framework ecosystem
- **3.4 THIRD-PARTY SERVICES** - External service integrations including KYC providers and blockchain infrastructure
- **6.2 DATABASE DESIGN** - Database architecture, schema design, and performance optimization strategies
- **6.5 MONITORING AND OBSERVABILITY** - Comprehensive monitoring infrastructure and observability patterns

- **8.1 DEPLOYMENT ENVIRONMENT** - Target environment specifications and infrastructure requirements