

ETL Plus Development Roadmap & Plans

Project Overview

ETL Plus is a comprehensive C++ backend application for Extract, Transform, Load operations with HTTP REST API, authentication, job management, and data transformation capabilities.

Current Status

Completed (Phase 1)

- ☒ Project structure and CMake build system
- ☒ Configuration management (JSON-based)
- ☒ Database manager foundation (PostgreSQL ready)
- ☒ HTTP server with Boost.Beast
- ☒ Authentication system (users, sessions, roles)
- ☒ ETL job manager with scheduling
- ☒ Data transformation engine
- ☒ REST API endpoints structure
- ☒ Multi-threaded architecture
- ☒ Basic compilation and execution

Known Issues

- ☒ **CRITICAL:** Segmentation fault in HTTP server when handling requests
- ☒ Config file copying not automated in CMake
- ☐ Database connections are simulated (not real PostgreSQL)
- ☐ No proper error handling for malformed JSON requests
- ☐ Memory management needs review

Development Phases

Phase 2: Core Stability & Bug Fixes (Week 1-2)

Priority: HIGH

2.1 Fix HTTP Server Issues

- ☐ Debug and fix segmentation fault in Boost.Beast implementation
- ☐ Add proper request/response lifecycle management
- ☐ Implement connection pooling
- ☐ Add request timeout handling
- ☐ Memory leak detection and fixes

2.2 Database Integration

- ☐ Integrate real PostgreSQL with libpqxx
- ☐ Implement connection pooling
- ☐ Add database schema creation scripts
- ☐ Transaction management improvements
- ☐ Database health checks

2.3 Enhanced Error Handling

- ☐ Structured error responses
- ☐ Input validation for all endpoints
- ☐ Proper exception handling
- ☐ Logging system implementation
- ☐ Request/response middleware

Phase 3: Feature Enhancement (Week 3-4)

Priority: MEDIUM

3.1 Advanced Authentication

- ☐ JWT token implementation
- ☐ Password hashing with bcrypt
- ☐ OAuth2 integration
- ☐ API key authentication
- ☐ Session persistence in database

3.2 Data Connectors

- ☐ CSV file reader/writer
- ☐ JSON data processor
- ☐ XML parser integration
- ☐ REST API data source connector
- ☐ Database-to-database connectors
- ☐ File system monitoring

3.3 Enhanced ETL Pipeline

- ☐ Visual pipeline builder (config-based)
- ☐ Data validation rules engine
- ☐ Custom transformation plugins
- ☐ Parallel processing capabilities
- ☐ Pipeline versioning
- ☐ Rollback mechanisms

Phase 4: Production Features (Week 5-6)

Priority: MEDIUM

4.1 Monitoring & Observability

- ☐ Prometheus metrics integration
- ☐ Health check endpoints expansion
- ☐ Performance monitoring
- ☐ Job execution statistics
- ☐ Resource usage tracking
- ☐ Alerting system

4.2 Web Dashboard

- ☐ React/Vue.js frontend
- ☐ Job management interface
- ☐ Real-time monitoring
- ☐ Pipeline visualization
- ☐ User management UI
- ☐ Configuration management

4.3 Security Hardening

- ☐ HTTPS/TLS support
- ☐ Rate limiting
- ☐ Input sanitization
- ☐ Security headers
- ☐ Audit logging
- ☐ Vulnerability scanning

Phase 5: Deployment & Scaling (Week 7-8)

Priority: LOW

5.1 Containerization

- ☐ Docker containerization
- ☐ Docker Compose setup
- ☐ Kubernetes manifests
- ☐ Helm charts
- ☐ Multi-stage builds

5.2 CI/CD Pipeline

- ☐ GitHub Actions workflows
- ☐ Automated testing
- ☐ Code quality checks
- ☐ Security scans
- ☐ Automated deployments

5.3 Documentation

- ☐ API documentation (OpenAPI/Swagger)
- ☐ User manual
- ☐ Developer guide
- ☐ Deployment guide
- ☐ Architecture documentation

Technical Debt & Improvements

Code Quality

- ☐ Unit test coverage (target: 80%+)
- ☐ Integration tests
- ☐ Code style enforcement (clang-format)
- ☐ Static analysis (cppcheck, clang-tidy)
- ☐ Memory profiling (valgrind)

Performance Optimization

- ☐ Request handling optimization
- ☐ Memory usage optimization
- ☐ Database query optimization
- ☐ Caching mechanisms
- ☐ Load testing

Architecture Improvements

- ☐ Plugin architecture for data sources
- ☐ Event-driven architecture
- ☐ Microservices decomposition
- ☐ Message queue integration (RabbitMQ/Kafka)
- ☐ Distributed processing support



Success Metrics

Phase 2 Goals

- Zero segmentation faults
- 100% uptime during testing
- Real database integration working
- All API endpoints responding correctly

Phase 3 Goals

- Support for 5+ data source types
- JWT authentication working
- Pipeline execution time < 10s for 1MB data

- Web dashboard functional

Phase 4 Goals

- Production-ready deployment
- 99.9% uptime
- Comprehensive monitoring
- Security audit passed

Phase 5 Goals

- Kubernetes deployment ready
- Full CI/CD pipeline
- Complete documentation
- Open source release ready



Development Environment Setup

Requirements

- C++20 compiler (GCC 10+ or Clang 12+)
- CMake 3.16+
- Boost 1.70+
- PostgreSQL 12+
- Docker (for containerization)
- Node.js 16+ (for frontend)

Development Workflow

1. Feature branch development
2. Unit tests required
3. Code review process
4. Integration testing
5. Performance validation
6. Documentation update



Decision Log

Technology Choices

- **C++20**: Modern features, performance
- **Boost.Beast**: HTTP server library
- **PostgreSQL**: Primary database
- **CMake**: Build system
- **JWT**: Authentication tokens
- **React**: Frontend framework

Architecture Decisions

- RESTful API design
- Multi-threaded job processing
- Plugin-based data connectors
- Configuration-driven pipelines
- Microservices-ready design

Iteration Process

Weekly Sprints

- Monday: Sprint planning
- Tuesday-Thursday: Development
- Friday: Testing & review
- Weekend: Documentation

Reviews

- Code review for all changes
- Architecture review for major features
- Security review for auth changes
- Performance review for core components

Support & Maintenance

Post-Release

- Bug fix releases (monthly)
- Feature releases (quarterly)
- Security updates (as needed)
- Documentation updates (ongoing)
- Community support (GitHub issues)

Last Updated: August 8, 2025

Version: 1.0

Status: Active Development