NADC Migration Project Summary - Updated

Project Overview

Background

The NADC (North American Data Center) migration involves moving infrastructure from RHEL 7 to RHEL 9 as part of a datacenter relocation. The primary challenge is that existing Perl scripts rely heavily on CPAN modules that are not available on the locked-down RHEL 9 environment.

Core Problem

- **Current State:** ControlM jobs invoke Perl scripts that use external CPAN dependencies
- **Target State:** RHEL 9 environment with no external CPAN installation permissions
- **Constraint:** ControlM job definitions must remain unchanged
- **Solution:** Create a hybrid Perl-Python architecture where Python handles CPAN functionality

Strategic Approach

Architecture Decision

Instead of rewriting all Perl scripts or changing ControlM jobs, we implement a "bridge" pattern:

1. ControlM continues to invoke existing Perl scripts (no job changes required)
2. Perl scripts become lightweight wrappers that delegate to Python
3. Python handles the heavy lifting using built-in or easily installable modules
4. The interface remains identical from ControlM's perspective

Key Benefits

- **Zero ControlM Changes:** Job definitions remain unchanged
- **Minimal Code Changes:** Only `use` statements need modification
- **Maintains Compatibility:** Same APIs, return values, and error handling
- **Future Proof:** Easier to maintain and extend Python codebase

CPAN Dependencies Inventory

✅ Completed Modules (5 of 11 - 45%)

Database Operations

- **DBI:** Core database interface (Oracle, Informix)
- **Replacement:** DBIHelper.pm + helpers/database.py
- **Status:** ✅ Complete and tested

Email Operations

- **Mail::Sender:** Email with attachments via SMTP
- **Replacement:** MailHelper.pm + helpers/email.py
- **Status:** ✅ Complete and tested

XML Processing

- **XML::Simple:** Basic XML parsing and manipulation
- **Replacement:** XMLHelper.pm + helpers/xml.py
- **Status:** ✅ Complete and tested

Date/Time Handling

- **Date::Parse:** String-to-date parsing
- **Replacement:** DateHelper.pm + helpers/dates.py
- **Status:** ✅ Complete and tested

Web/Network Operations

- **LWP::UserAgent:** HTTP client operations
- **WWW::Mechanize:** Web form automation (simple patterns)
- **Replacement:** HTTPHelper.pm + helpers/http.py
- **Status:** ✅ Complete and tested (single module handles both)

📋 Pending Modules (6 remaining)

File Operations

- **Excel::Writer::XLSX:** Excel file generation
- **Status:** Priority 1 - Next implementation target

Network Operations

- **Net::SFTP::Foreign:** SFTP file transfers
- **Status:** Priority 2

Cryptography & Security

- **Crypt::CBC:** Block cipher operations
- **Crypt::SSLeay:** SSL/TLS support
- **Status:** Priority 3

Utilities

- **Log::Log4perl:** Advanced logging
- **XML::XPath:** XPath query support (potential XMLHelper extension)
- **Status:** Priority 3

Technical Implementation

Core Infrastructure Components

CPANBridge.pm (v1.01)

- Base class for all CPAN replacements
- Handles Perl-to-Python communication via JSON over pipes
- Manages error handling, timeouts, and retry logic
- Provides debugging and performance monitoring
- Platform-specific optimizations (Windows vs Linux)

cpan_bridge.py (v1.0.1)

- Python bridge script that receives and routes requests
- Validates input for security
- Loads and manages helper modules dynamically
- Returns structured JSON responses
- Includes comprehensive error handling and logging

Helper Module Pattern

Each CPAN module replacement follows this structure:

- **Perl Wrapper** (e.g., DBIHelper.pm, MailHelper.pm, HTTPHelper.pm): Provides identical API to original CPAN module
- **Python Implementation** (e.g., helpers/database.py, helpers/email.py, helpers/http.py): Contains actual functionality

- **Bridge Communication:** JSON-based request/response between layers

Communication Flow

```
ControlM Job → Perl Script → CPAN Replacement → CPANBridge → Python Helper → Database/Service
```

Current Status

✅ Completed Components

Infrastructure

1. **CPANBridge Infrastructure:** Core communication layer working on both Windows and RHEL 9
2. **Error Handling:** Comprehensive error management and retry logic
3. **Platform Compatibility:** Windows (development) and RHEL 9 (target) validated

Module Replacements

1. **DBI Replacement (DBIHelper.pm)**
   - Complete drop-in replacement for database operations
   - Supports Oracle and Informix connections
   - All standard DBI methods implemented
   - Transaction support included
   - Error handling compatible with existing code
2. **Mail::Sender Replacement (MailHelper.pm)**
   - Complete API compatibility including method chaining
   - Supports email attachments with various encodings
   - Works with localhost SMTP (no authentication)
   - Maintains compatibility variables ($Mail::Sender::NO_X_MAILER, $Mail::Sender::Error)
   - Tested with exact usage pattern from mi_email_resultset.pl
3. **XML::Simple Replacement (XMLHelper.pm)**
   - Full XMLin/XMLout API compatibility
   - Python xml.etree.ElementTree backend
   - Handles all XML::Simple options and configurations
   - Maintains data structure compatibility
4. **Date::Parse Replacement (DateHelper.pm)**

- String-to-date parsing functionality
- Python datetime backend
- Timezone handling support
- Compatible with enterprise date formats

5. **LWP::UserAgent + WWW::Mechanize Replacement (HTTPHelper.pm)**
   - Single module handles both LWP::UserAgent and WWW::Mechanize
   - Complete HTTP::Request compatibility
   - SSL verification control via environment variables
   - Form-encoded POST handling
   - WebSphere server health checking (WWW::Mechanize pattern)
   - Python urllib backend using only standard library

🔄 Validated Functionality

- Perl-to-Python communication bridge operational
- JSON serialization/deserialization working
- Windows pipe communication issues resolved (file-based approach)
- Linux IPC::Open3 implementation working
- Database connection framework tested
- Email operations with attachments validated
- XML processing end-to-end tested
- HTTP operations including form submissions working
- Performance overhead measured at 2-4ms per operation

Progress Metrics

Overall Statistics

- **Modules Complete:** 5 of 11 (45%)
- **Core Infrastructure:** 100% complete
- **Performance Overhead:** 2-4ms per operation (<1% of typical operation time)
- **API Compatibility:** 100% for completed modules
- **ControlM Job Changes Required:** 0

Implementation Velocity

- **Phase 1 (Infrastructure):** 4 weeks - Complete
- **Phase 2 (First 2 modules):** 3 weeks - Complete
- **Phase 3 (Next 3 modules):** 2 weeks - Complete
- **Current Rate:** ~1.5 modules per week

Risk Assessment

✅ Mitigated Risks

- Bridge architecture proven stable and reliable
- Performance overhead acceptable for enterprise use
- API compatibility achieved for complex modules
- Cross-platform compatibility validated
- Major CPAN dependencies successfully replaced

⚠️ Remaining Challenges

- Excel file format complexity (XLSX generation)
- SFTP authentication methods and key management
- Cryptographic operations requiring specialized libraries
- Production deployment and rollback procedures

Next Steps & Timeline

Immediate Priorities (Next 4 weeks)

Week 1-2: File Operations

- Excel::Writer::XLSX implementation
- Python openpyxl or xlsxwriter backend evaluation
- Workbook/worksheet compatibility testing
- Format preservation validation

Week 3-4: Network Operations

- Net::SFTP::Foreign implementation
- Python paramiko vs built-in approaches
- SSH key authentication handling

- Secure file transfer validation

Remaining Work (Weeks 5-8)

Security & Logging

- Crypt::CBC implementation for encryption operations
- Log::Log4perl replacement for advanced logging
- XML::XPath extension (if required by analysis)
- Final integration and stress testing

Success Criteria

- All 11 CPAN modules replaced with 100% API compatibility
- Zero changes required to ControlM job definitions
- Performance overhead remains under 5ms per operation
- Full backwards compatibility with existing Perl business logic
- Comprehensive test coverage for all replacement modules

Deployment Strategy

Pilot Phase

- Deploy completed modules (5) to development environment
- Validate with subset of ControlM jobs
- Performance monitoring and optimization
- User acceptance testing

Production Rollout

- Phased migration by application/job type
- Parallel running during transition period
- Monitoring and rollback procedures
- Documentation and training completion

Estimated Completion

- **All modules complete:** 6-8 weeks from current date
- **Pilot deployment:** 8-10 weeks
- **Full production migration:** 12-14 weeks

The project demonstrates significant progress with nearly half of all required CPAN modules successfully replaced while maintaining the core principle of zero changes to existing ControlM job definitions.