4.0 - Tasks: As-Built Report Generator

Initial Development Tasks: VAST As-Built Report Generator

Version: 2.0

Created: 2025-09-11

Overview

This document outlines the initial development tasks for the VAST Data As-Built Report Generator project, organized according to the 2-sprint Agile methodology defined in the Project Plan. Each task is designed to be granular, measurable, and aligned with the project's core objectives.

Sprint 1: Core Functionality & Data Collection (Weeks 1-2)

Phase 1.1: Project Setup and Infrastructure

- [x] **Task 1.1.1: Repository Initialization**
  - Clone the GitHub repository: `git@github.com:rstamps01/ps-deploy-report.git`
  - Create the complete directory structure as defined in the development guide
  - Initialize Git workflow with `main`, `develop`, and feature branches
  - Create initial `.gitignore` file for Python projects
  - **Deliverable:** Properly structured repository with all required directories
- [x] **Task 1.1.2: Development Environment Setup**
  - Create `requirements.txt` with initial dependencies (requests, pyyaml, reportlab, pytest)
  - Set up Python virtual environment
  - Create basic configuration file template (`config/config.yaml`)
  - **Deliverable:** Functional development environment ready for coding
- [ ] **Task 1.1.3: Logging Infrastructure**
  - Implement logging configuration in `src/utils.py`

- Create log directory structure and rotation mechanism
- Define log message formats and levels as per the development guide
- **Deliverable:** Centralized logging system ready for use across all modules

Phase 1.2: API Handler Module Development

☐ **Task 1.2.1: Basic API Client**

- Create `src/api_handler.py` with basic HTTP client functionality
- Implement connection management and timeout handling
- Add basic error handling for network failures
- **Deliverable:** Functional API client that can make HTTP requests

☐ **Task 1.2.2: VAST API Authentication**

- Implement session-based authentication for VAST API
- Add credential management (environment variables, secure prompts)
- Implement session cookie handling and storage
- **Deliverable:** Secure authentication system for VAST clusters

☐ **Task 1.2.3: Retry Logic and Fault Tolerance**

- Implement exponential backoff retry mechanism
- Add handling for specific HTTP status codes (401, 404, 500, etc.)
- Create graceful degradation for non-critical API failures
- **Deliverable:** Resilient API client that handles transient failures

Phase 1.3: Data Extraction Module Development

☐ **Task 1.3.1: Core Data Extraction Functions**

- Create `src/data_extractor.py` with functions for each report section
- Implement `get_cluster_info()` for executive summary data
- Implement `get_hardware_inventory()` for CNode/DNode details
- **Deliverable:** Functions to extract cluster overview and hardware data

☐ **Task 1.3.2: Network Configuration Extraction**

- Implement `get_network_config()` for DNS, NTP, and VIP pool data
- Add data validation and error handling for missing network configurations
- **Deliverable:** Complete network configuration data extraction

☐ **Task 1.3.3: Logical Configuration Extraction**

- Implement `get_logical_config()` for tenants, views, and view policies

- Implement `get_security_config()` for authentication providers
- Implement `get_data_protection_config()` for snapshot policies
- **Deliverable:** Complete logical configuration and security data extraction

Phase 1.4: JSON Report Generation

☐ **Task 1.4.1: Data Aggregation and Structuring**

- Create data aggregation functions in `src/report_builder.py`
- Define JSON schema for the complete report structure
- Implement data validation and sanitization
- **Deliverable:** Structured JSON output with all collected data

☐ **Task 1.4.2: CLI Interface Development**

- Create `src/main.py` with command-line argument parsing
- Implement configuration file loading and validation
- Add help documentation and usage examples
- **Deliverable:** Functional CLI tool that generates JSON reports

☐ **Task 1.4.3: End-to-End Testing and Validation**

- Test complete workflow against a live VAST cluster
- Validate JSON output structure and completeness
- Document any API inconsistencies or missing data scenarios
- **Deliverable:** Validated JSON report generation capability

---

Sprint 2: Report Formatting & Finalization (Weeks 3-4)

Phase 2.1: PDF Report Generation

☐ **Task 2.1.1: PDF Template Design**

- Create HTML template in `templates/report_template.html`
- Design professional layout with title page, table of contents, and sections
- Implement responsive formatting for various data sizes
- **Deliverable:** Professional PDF template ready for data injection

☐ **Task 2.1.2: PDF Generation Engine**

- Implement PDF generation functions using reportlab or weasyprint
- Add data formatting and presentation logic
- Implement error handling for PDF generation failures

- **Deliverable:** Functional PDF report generation from JSON data

☐ **Task 2.1.3: Report Formatting and Styling**

- Apply professional styling and branding to PDF output
- Implement tables, charts, and visual elements for data presentation
- Add page numbering, headers, and footers
- **Deliverable:** Professionally formatted PDF reports

Phase 2.2: Testing and Quality Assurance

☐ **Task 2.2.1: Unit Test Development**

- Create comprehensive unit tests in `tests/` directory
- Implement mock API responses for testing data extraction
- Achieve minimum 80% code coverage
- **Deliverable:** Complete unit test suite with high coverage

☐ **Task 2.2.2: Integration Testing**

- Develop end-to-end integration tests
- Test error scenarios and edge cases
- Validate report accuracy against known cluster configurations
- **Deliverable:** Validated integration test suite

☐ **Task 2.2.3: Performance Optimization**

- Profile application performance and identify bottlenecks
- Implement concurrent API calls where appropriate
- Optimize data processing and report generation
- **Deliverable:** Optimized application meeting performance requirements

Phase 2.3: Documentation and Packaging

☐ **Task 2.3.1: User Documentation**

- Create comprehensive `README.md` with installation and usage instructions
- Document configuration options and troubleshooting steps
- Add example usage scenarios and sample outputs
- **Deliverable:** Complete user documentation

☐ **Task 2.3.2: Code Documentation**

- Add comprehensive docstrings to all functions and classes
- Create inline comments for complex logic
- Generate API documentation if needed

- **Deliverable:** Fully documented codebase

☐ **Task 2.3.3: Final Packaging and Release Preparation**

- Finalize `requirements.txt` with exact version dependencies
- Create installation and deployment scripts
- Prepare release notes and version tagging
- **Deliverable:** Production-ready package

Success Metrics

Each task completion will be measured against these criteria:

1. **Functionality:** Code executes without errors and produces expected output
2. **Code Quality:** Adheres to PEP 8 standards and development guide requirements
3. **Testing:** Includes appropriate unit tests with passing results
4. **Documentation:** Contains clear docstrings and comments
5. **Git Workflow:** Properly committed with descriptive messages and pushed to repository

---

Risk Mitigation

High-Priority Risks:

- **API Access Issues:** Ensure early access to test VAST cluster
- **Authentication Complexity:** Implement robust credential handling from the start
- **Data Structure Variations:** Build flexible parsing with graceful error handling

Contingency Plans:

- Mock API responses for offline development if cluster access is limited
- Modular design allows for easy updates if API endpoints change
- Comprehensive logging enables quick troubleshooting of field issues

---

This task outline provides a clear roadmap for the development effort while maintaining flexibility for adjustments based on discoveries during implementation.

Updated Tasks for Additional API Data Points (September 2025)

Updated Task 1.2.1: Basic API Client (Enhanced)

Additional Requirements:

- Include support for rack height data collection via `Schema/CBox/index_in_rack` and `Schema/DBox/index_in_rack`
- Add PSNT data collection via `Schema/Cluster/psnt`
- Update API endpoint mapping to include these new data points
- **Updated Deliverable:** Enhanced API client with improved data collection coverage (~80% automated)

Updated Task 1.3.1: Core Data Extraction Functions (Enhanced)

Additional Requirements:

- Update `get_cluster_info()` to include PSNT field extraction
- Update `get_hardware_inventory()` to include rack height information for CBoxes and DBoxes
- Add validation for rack height data format and PSNT format
- **Updated Deliverable:** Enhanced data extraction with physical positioning and support tracking data

New Task 1.3.4: Enhanced Hardware Positioning

Requirements:

- Implement `get_rack_positions()` function for physical hardware layout
- Create data mapping between logical hardware IDs and physical rack positions
- Add validation for rack height data consistency
- Implement fallback handling for clusters without rack position data
- **Deliverable:** Complete physical hardware positioning data collection

Updated Task 1.4.1: Data Aggregation and Structuring (Enhanced)

Additional Requirements:

- Update JSON schema to include rack height fields in hardware sections
- Add PSNT field to cluster identification section
- Implement data validation for new fields
- Update data sanitization to handle rack position formats
- **Updated Deliverable:** Enhanced JSON structure with improved hardware and cluster identification data

Updated Task 1.4.3: End-to-End Testing and Validation (Enhanced)

Additional Requirements:

- Test rack height data accuracy against physical installation documentation
- Validate PSNT format compatibility with VAST support systems
- Test graceful handling of missing rack height or PSNT data
- Document any version compatibility issues with older VAST clusters
- **Updated Deliverable:** Comprehensive validation including new API data points

---

Implementation Notes for Enhanced API Coverage

Priority Updates

1. **High Priority:** Update API handler to collect rack heights and PSNT data
2. **Medium Priority:** Enhance report formatting to display physical layout information
3. **Low Priority:** Add advanced rack visualization features

Backward Compatibility

- Ensure the tool works with VAST clusters that may not have rack height data
- Implement version detection to handle API schema differences
- Provide clear error messages when enhanced data is not available

Testing Strategy

- Test against multiple VAST cluster versions (5.1, 5.2, 5.3)
- Validate data accuracy with physical installation records
- Test error handling for partial data availability