LVIC Integration Automation

Status	In progress
Project	LVIC Integration Automation
Sub-tasks	Automation Integration Docker POC
Tags	

Summary

Project Title: Integration Automation System

Technologies Involved:

- **Front-End:** React for building a single-page application (SPA) without a traditional home page, offering a project portal and an automation hub.
- **Back-End:** Python with Flask framework, Postgres for database management, Ollama AI for automation, and Ansible for configuration management and provisioning.

Process Overview:

1. **User Portal Access:** Users log into the web portal to start a new project.
2. **Document Processing:** Users upload Excel or other documents which are automatically parsed by AI. The AI fills in form fields based on document content.
3. **Validation:** Human users validate the form entries and submit them for processing.
4. **Data Management:** Submitted data is stored in the database indexed by project ID.

5. **Network Configuration:** Applied switches are Zero-Touch Provisioned (ZTP) and network inventory is built using Ansible and Python. These tools also provision necessary network settings.
6. **Infrastructure Deployment:** Ansible is used to deploy and configure virtual infrastructure through vCenter.
7. **Documentation:** Delivery documentation is automatically generated using a combination of AI, Python, and Ansible.

Outcome:

The Integration Automation System streamlines new project setup, document processing, network provisioning, and documentation, significantly reducing manual tasks and increasing efficiency and accuracy within business processes.

1. Front-End Setup

- **Structure**: Use a React single-page application (SPA) setup, structured into modular components for reusability.
- **Core Components**:
  - **Login/Authentication**: Secure user authentication and session handling.
  - **Project Dashboard**: Displays a list of active projects and allows users to initiate new projects.
  - **File Upload & Form Validation**: Form components to handle file uploads (Excel, etc.) and display parsed data for human validation.
  - **Automation Hub**: A user-friendly interface where automation tasks (ZTP, Ansible deployment) can be initiated or monitored.
- **Requirements**:
  - `npx create-next-app@latest` as per your current setup.
  - Integration with backend API endpoints (Python/Flask).
  - Error handling and validation for user input and document parsing.

- **Tech**: React, JavaScript, CSS modules or styled-components for styling, and Next.js for SSR support (if needed).

2. Back-End API Setup (Python + Flask)

- **Structure**:
  - **API Endpoints**: Set up RESTful endpoints in Flask to handle CRUD operations for projects, document uploads, and automation tasks.
  - **Database**: Use PostgreSQL to store project data, document metadata, parsed form data, etc.
  - **AI Parsing (Ollama)**: Integrate an AI model to parse Excel/CSV files, fill in initial form fields, and save parsed data.
- **Core Features**:
  - **Document Processing API**: Endpoint to receive uploaded documents, parse them using AI, and return structured data.
  - **Project Management**: CRUD endpoints for managing project data and linking documents and configurations.
  - **Data Validation**: Validation layer to ensure data consistency before storing it in the database.
- **Requirements**:
  - Install necessary Python packages (Flask, SQLAlchemy, psycopg2 for PostgreSQL integration).
  - Connect the API to the frontend, ensuring CORS policies are handled correctly.
  - Configure environment variables for sensitive data (API keys, database credentials).
- **Tech**: Python, Flask, SQLAlchemy (ORM), PostgreSQL.

3. Database Structure

- **Core Tables**:

    - **Projects**: To store general information about each project (e.g., name, ID, created date).
    - **Documents**: Linked to projects, stores metadata for uploaded files.
    - **Parsed Data**: Structured data extracted from documents (VLANs, server info, shipping info, etc.).
    - **Network Inventory**: Stores data about network configurations and devices.
- **Conditional Fields**: Use JSONB fields for flexible storage of configuration data like VLANs and VSANs, which vary between projects.
- **Relationships**:
    - **One-to-Many** between Projects and Documents.
    - **One-to-Many** between Projects and Parsed Data entries.
    - **One-to-Many** between Projects and Network Inventory (to track devices and configurations).
- **Schema Management**: Use Alembic or another migration tool to version and manage schema changes over time.

4. ZTP Process Setup

- **Purpose**: Zero Touch Provisioning (ZTP) automates initial switch setup, configuring Cisco and Dell switches with a base configuration.
- **Structure**:
    - **DHCP Server**: Set up a DHCP server in the lab environment to assign IP addresses to new switches.
    - **TFTP Server**: Store configuration scripts for default switch setup; DHCP server will direct switches to fetch configurations from the TFTP server.
- **Scripts and Configurations**:
    - **DHCP Configurations**: Map MAC addresses to specific configurations (if required).

    - **Base Configurations**: Scripts for each switch type (Dell, Cisco) that define initial VLANs, IPs, and SNMP configurations.
- **Automation Tools**: Docker Compose setup for prototyping and local testing. Use Ansible to push further configurations after ZTP.
- **Testing and Deployment**: Test ZTP in a local Docker environment and deploy to the lab with vCenter when stable.

5. Ansible Configuration and Network Provisioning

- **Structure**:
    - **Inventory Files**: Dynamically generated by Python scripts based on parsed project data.
    - **Playbooks**:
        - **Network Provisioning**: Playbooks to set up VLANs, IP schemes, and DNS settings.
        - **vCenter Deployment**: Configure vCenter servers, VMs, and network connections.
    - **Templates**: Use Jinja2 templates for flexible configuration files, which adapt to each project's data.
- **Process**:
    - **Generate Inventory**: Parse project data and generate an inventory file for Ansible to read.
    - **Run Playbooks**: Execute playbooks to apply configurations to network devices, switches, and vCenter.
    - **Status and Logs**: Log success/failure for each step and make logs accessible in the automation hub.
- **Requirements**:
    - Ansible installed with necessary plugins for network and VMware automation.
    - Access to devices in the lab environment for testing.

6. Delivery Documentation Automation

- **Goal**: Automatically generate and format delivery documentation after deployment is complete.
- **Structure**:
  - **Data Collection**: Collect data from each deployment step, including network configurations, device lists, and final project settings.
  - **Documentation Template**: Create templates with Jinja2 or similar to format delivery documents.
  - **Generation Process**:
    - **Script Execution**: Execute Python/Ansible scripts to pull data from the database and Ansible logs.
    - **Template Filling**: Use the data to fill in delivery templates, including network topology, device settings, and final configurations.
    - **File Export**: Export documentation as PDF or Word files, linked to the project in the web portal.
- **Tech**: Python, Jinja2 templates, and possibly a PDF generation library (like ReportLab or WeasyPrint).

7. Testing and Deployment

- **Local Testing**: Test each module (frontend, backend, Ansible scripts, ZTP) independently in a Docker environment.
- **Integration Testing**: Run end-to-end tests to ensure that uploaded documents can be parsed, validated, and fed into the automation workflow.
- **Lab Deployment**: Deploy the solution to the lab environment with vCenter and connected devices.
- **Continuous Integration/Deployment (CI/CD)**:
  - Use GitLab CI, GitHub Actions, or Jenkins for automatic deployment to test environments.
  - Set up automated tests to validate each deployment phase before release to the lab environment.

8. Project Documentation and Training

- **Documentation**:
  - **User Guides**: Instructions for using the web portal, uploading documents, and validating AI-parsed data.
  - **Developer Documentation**: For future development and troubleshooting, including API documentation, ZTP setup, and Ansible configuration notes.
- **Training**:
  - **Portal Training**: Brief training for end-users on how to initiate and monitor projects.
  - **Admin Training**: For IT admins managing ZTP, Ansible, and vCenter configurations.