## Introduction

The Custom NiFi-Inspired ETL Platform is a comprehensive, enterprise-level data integration and processing tool developed using Java 17 and Spring Boot 3. The aim of the project is to replicate the core capabilities of Apache NiFi while leveraging the Java and Spring ecosystem for better modularity, security, and DevOps integration. The system enables users to design, execute, and monitor Extract, Transform, Load (ETL) workflows dynamically, allowing for real-time data processing, data migration, and integration between heterogeneous systems.   
  
ETL workflows in this platform consist of processors connected in a flow-based programming style. Each processor is developed in Java is responsible for a specific action, such as extracting data from a MySQL database, applying transformation rules, or loading data into AWS S3. The platform supports pluggable processors, enabling developers to extend functionality without modifying the core system.  
  
**Key features include**:

1. Dynamic creation and modification of ETL workflows via RESTful APIs.  
2. Pluggable processor framework based on Java, Spring for extending the platform with new data sources, transformations, and sinks.  
3. High-performance execution using Java concurrency APIs for parallel processing.  
4. Security-first approach with Spring Security and JWT authentication.  
5. Deployment-ready architecture for AWS EKS with integrated CI/CD pipelines.  
6. Observability using Spring Boot Actuator, Micrometer, Prometheus, and Grafana.

## Architecture

Apache NiFi is designed to automate the flow of data between systems. Its architecture centers around a Java Virtual Machine (JVM) running on a host operating system, comprising several key components that work together seamlessly.

1. Web Server

The Web Server in NiFi is responsible for hosting the HTTP-based command and control API. This component provides the user interface and is the gateway for managing and monitoring data flows. Users can interact with NiFi through the web server, configure processors, and monitor the overall system.

1. Flow Controller

The Flow Controller is the central intelligence of NiFi. It manages threads for extensions to run on and schedules when these extensions receive resources to execute. It plays a crucial role in orchestrating the flow of data and ensuring that various components operate in a coordinated manner.

1. Extensions

NiFi supports various types of extensions, each serving a specific purpose. These extensions operate and execute within the JVM, enhancing the functionality of NiFi. The specific types of extensions are described in separate documents, and they contribute to the flexibility and extensibility of the NiFi platform.

1. FlowFile Repository

The FlowFile Repository is where NiFi keeps track of active FlowFiles within the data flow. It is pluggable, allowing for different implementations. The default approach involves a persistent Write-Ahead Log on a specified disk partition. This repository ensures the resiliency and durability of FlowFile information.

1. Content Repository

The Content Repository stores the actual content bytes of a given FlowFile. Similar to the FlowFile Repository, its implementation is pluggable, with the default approach utilizing a file system mechanism to store data blocks. Multiple file system storage locations can be specified to distribute storage and reduce contention on any single volume.

1. Provenance Repository

The Provenance Repository is where all provenance event data is stored. This repository is pluggable, and the default implementation involves using one or more physical disk volumes. It indexes and makes event data searchable, providing insights into the history and lineage of data flows.

## Apache NiFi Tutorial: What is NiFi? Architecture & Installation

## Technologies Used

### Back End Technology

Spring Boot is at the core of the backend, providing rapid application development, embedded servers, and minimal configuration overhead. The backend is composed of several Spring Boot modules:  
- Spring Web: For creating RESTful APIs to interact with the ETL workflows.  
- Spring Data JPA: For seamless integration with MySQL to persist flow definitions and execution logs.  
- Spring Security: To secure the platform with JWT-based authentication and authorization.  
- Spring Boot Actuator: For exposing health checks and operational metrics.

Maven is used as the build automation and dependency management tool, ensuring consistency across environments.

### Database

The platform uses MySQL, deployed on AWS RDS, to store workflow definitions, processor metadata, and provenance events. The schema is designed to maintain referential integrity between flows, processors, and their relationships. Indexes are created for performance-critical queries, and automated backups ensure disaster recovery.

### Unit Testing

JUnit 5 and Mockito are employed for unit testing and mocking dependencies. Test cases cover all business logic, with a focus on achieving at least 85% code coverage. Integration tests are run using Testcontainers to spin up ephemeral MySQL and Redis instances during the test lifecycle. All tests are integrated into a Jenkins CI pipeline, ensuring they are executed automatically on code changes.

### Deployment

The application is containerized using Docker, with multi-stage builds for optimized image sizes. Deployment is handled via Kubernetes manifests. **Terraform is used to provision AWS EKS, RDS, and S3 resources, while Ansible manages application configuration**.  
  
Deploying applications using Continuous Integration/Continuous Deployment (CI/CD) pipelines and AWS cloud services, ensuring efficient, scalable, and secure deployment processes.

* Utilized automated CI/CD pipelines, reducing deployment times and ensuring consistent, reliable releases.
* Deployed scalable and high-availability applications on AWS EC2, leveraging Auto Scaling and ELB for optimal performance.
* Utilized AWS S3 for efficient storage and management of static assets and backups, ensuring data durability and availability.
* Managed secure access to AWS resources using IAM, enforcing strict access controls and enhancing security posture.
* Enabled decoupled communication and timely notifications using AWS SNS, improving system monitoring and alerting capabilities.

### Security

Security is implemented using Spring Security with JWT tokens for stateless authentication. Role-based access control ensures that only authorized users can modify or execute certain workflows. Sensitive credentials are stored securely using Kubernetes Secrets and AWS IAM Roles for Service Accounts (IRSA).

## Methodology

The project follows the Agile methodology, with development carried out in iterative sprints. Each sprint includes requirements gathering, design, development, testing, and deployment phases. Daily stand-ups and sprint retrospectives ensure continuous improvement and alignment with project goals.

