Apache Nifi

Automação e Integração de Dados

Pedro Saito e Milton Salgado

Laboratório de Métodos de Suporte à Tomada de Decisão

Universidade Federal do Rio de Janeiro

2025-07-27

LAMDEC

Sumário

1. Introdução

LAMDEC

1.1 História

Contexto: O Apache Nifi foi inicialmente desenvolvido pela **NSA** (*National Security Agency*) em 2006, anteriormente conhecido por *"NiagaraFiles"*, para ser uma ferramenta de **automação de fluxo de dados**.

Em 2014, o projeto foi incorporado pelo **Apache Software Foundation** como um projeto *Open Source*.

Baseado no paradigma **FBP** (*Flow-Based Programming*).

★ 5.4k **Github** 510 **Contribuintes** ☕ **Java 21**

1.2 Definição

Ferramenta *low-code* de automação de fluxo de dados em tempo real (*streaming data*).

Plataforma primária: Aplicação Web **Nativa (Java 21)** ou **Docker**.

Casos de uso principais:

- Workflows moderados de ETL (*real time* ou *event-driven*).
- Processos interconectados e em constante mudança.
- Migração de dados entre sistemas legados e modernos
- Sincronização de bancos de dados em tempo real

1.3 Funcionalidades

- Gerenciamento de múltiplos usuários com diferentes permissões.
- **Data Lineage**: Monitoramento e rastreabilidade dos dados
- Subir uma instância no *Tailscale* para uso interno.
- **Tolerância a falhas** com retry automático e dead-letter queues.
- **Escalabilidade horizontal** com clustering gerenciado pelo ZooKeeper.

https://localhost:8443/nifi/

1.4 Automatização de Tarefas

O Apache NiFi automatiza diversas tarefas críticas:

Cibersegurança

Coleta de logs de segurança
Correlação de eventos suspeitos
Análise de tráfego real

Compliance e Auditoria

Rastreamento de dados sensíveis
Relatórios automáticos
Políticas de retenção
Conformidade LGPD/GDPR

Observabilidade

Métricas agregadas
Logs em tempo real
Dashboards automáticos

Detecção de Anomalias

Análise de padrões em transações
Comportamentos anômalos
Integração com ML models
Alertas em tempo real

2. Componentes

LAMDEC

2.1 Terminologia

- ***Dataflow Manager*** : Usuário Nifi com permissões para adicionar, remover ou modificar os componentes do *Nifi Flow*.
- **FlowFile**: Unidade básica de dados. Composta pelos *Attributes* (metadados) e o *Content* (conteúdo). Principais metadados:
  - **UUID**: Identificador único universal de 128 bits (RFC 4122 do IETF).
  - **Filename**: Nome do arquivo/dado.
  - **Path**: Diretório estruturado para armazenamento no disco ou serviço externo.

2.1 Terminologia

- **Processor**: Componente lógico básico responsável pelas tarefas:
  - Exemplos de *Processors*: `GetFile`, `PutFile`, `ExecuteSQL`, `InvokeHTTP`, `ReplaceText`, `UpdateAttribute`, `LogAttribute`, `RouteOnAttribute`.

Ingestão **Transformação** **Roteamento**

- **Relacionamentos**: Cada processador pode ter zero ou mais relacionamentos definidos. Saída lógica do processador (**Success** ou **Failure**). Após o término do processamento, o FlowFile deve ser roteado/transferido para outro componente escolhido pelo DataFlow Manager.

2.1 Terminologia

- **Canvas**: Interface visual primária para o *design*, gerenciamento e monitoramento dos *dataflows*.
- **Conexão**: Componente "físico" que interliga quaisquer dois elementos do *dataflow* (sobretudo *Processors*). Ligação física entre 2 componentes do fluxo de dados.
- **Bulletin**: Os componentes podem reportar *bulletins*, isto é, equivalente a mensagens de *logs* com tempo e nível de severidade (*Debug*, *Info*, *Warning* e *Error*).

2.1 Terminologia

- **Controller Service**: Componente que provê funcionalidades e configurações para outros elementos (sobretudo *Processors*).
  - **Exemplo**: `DBCPConnectionPool` fornece serviço de pool de conexões para um ou mais processadores. Propriedades obrigatórias:
    - **Database Connection URL**: `jdbc:sqlserver://ip;port;db`
    - **Database Driver Class Name**: com.Microsoft.sqlserver.jdbc.SQL
    - **Driver's Path**: `~/mssql-drivers/ptb/jars`
    - **Database User & Password**: sa ****

2.1 Terminologia

- **Funnel**: Componente que permite combinar dados de diferentes conexões em uma só.
- **Process Group**: Agrupamento lógico de *Processors* para facilitar o gerenciamento do *dataflow*.
- **Remote Process Group**: Transferência de dados (*FlowFiles*) para uma instância remota do *Nifi*.
- **Reporting Tasks**: Tasks executadas no *background* que providenciam relatórios de utilização e monitoramento da instância *Nifi*.
  - Exemplos: `MonitorDiskUsage`, `MonitorMemory`.

3. Aplicação

LAMDEC

3.1 Exemplo

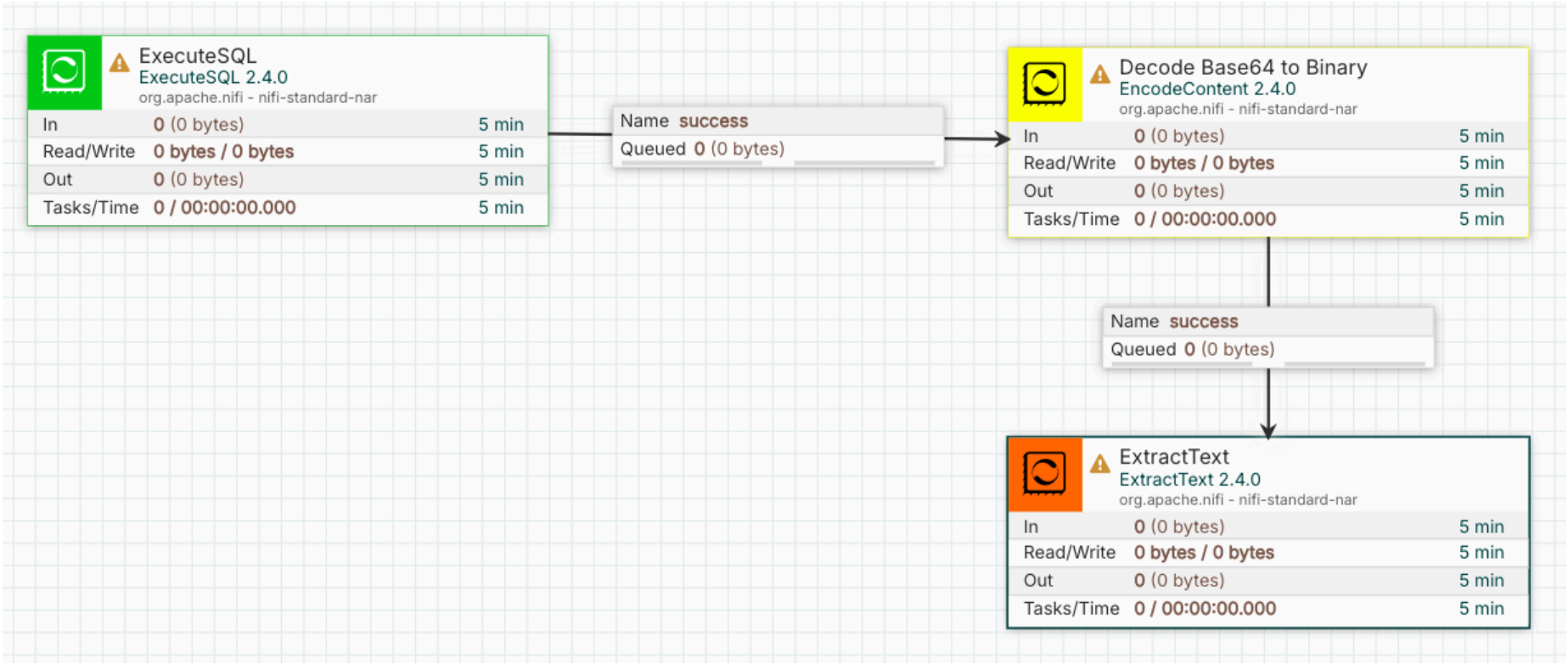
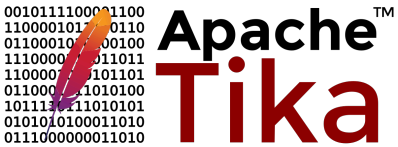


Figura 1: *Fluxo de consulta do `base64`, decodificação e extração de texto.*

3.1 Exemplo

```sql
SELECT TOP 10 base64encode
FROM [pgmconsultaprocesso].[dbo].[documentos]
WHERE
    sucesso = 1
AND ext = '.pdf'
```

Base64 → Binary



4. Códigos

LAMDEC

4.1 Scripts

Dois *Processors* primordiais para execução de comandos externos:

- **ExecuteScript**: Executa scripts em linguagens como Python, Groovy, JavaScript diretamente no processador.
- **ExecuteStreamCommand**: Passa o conteúdo do *FlowFile* como input, executa uma série de comandos arbitrários e retorna o resultado da transformação.

```python
import sys, json
from datetime import datetime
data = json.load(sys.stdin)
data['modificada_em'] = datetime.utcnow().isoformat()
json.dump(data, sys.stdout)
```

Listagem 1: *ExecuteStreamCommand para adicionar metadado datetime.*

4.2 *Custom Processors*

Na ausência de *Processors* que cumprem a tarefa desejada, pode-se definir seus próprios *Processors* em **Java** e importá-los no *dataflow*.

```java
public class ExampleProcessor extends AbstractProcessor {
  public static final Relationship REL_FAILURE = new Relationship
    .builder()
    .description("Failed processing")
    .name("failure")
    .build();

  public static final Relationship REL_SUCCESS = new Relationship
      .builder()
      .description("Succeed processing")
      .name("success")
      .build();

  @Override
  public Set<Relationship> getRelationships() {
      return new HashSet<>() {{
          add(REL_FAILURE);
          add(REL_SUCCESS);
      }};
  }
}
```

→ .Nar (Nifi Archive)

5. Comparações

LAMDEC

5.1 Airflow

Atributos	Apache Nifi	Apache Airflow
Interface	GUI *Drag-and-Drop*	Código Python (DAGs)
Escalabilidade	Cluster horizontal	Executors
Periodicidade	*Event-Driven*	Time-based schedules
Utilização	Interface Gráfica	Python *script*
Monitoramento	*Provenance events*	Logging + interface
Concorrência	Back-pressure, priorização, load--balancing entre nós	Execução paralela via Executors (Celery, Kubernetes, Local)

5.1 Airflow

Atributos	Apache Nifi	Apache Airflow
Data Handling	*FlowFiles* com conteúdo + atributos; streaming ou micro-batch	Task outputs persistidos (XComs, arquivos, databases); principalmente batch
Failure Handling	Retry automático, dead-letter queues, rastreamento de proveniência	Políticas de retry por task, alertas SLA, intervenção manual

5.2 Desvantagens

- **Reproducibilidade**: Dificuldade para copiar um *workflow* de uma instância do *Nifi* para outra. É preciso recriar cada componente manualmente.
- **Dinamidicidade**: Dificuldade na modelagem de *dataflows* que se ajustam constantemente (mais complexos).
  - Exemplo: SQL Dinâmico.
- **Depuração**: Herda características do paradigma **FBP** que concebem os *Processadores* como caixas pretas. Em algum momento será preciso olhar por de trás dos panos.

5.3 Vantagens

- **Facilidade no Uso**: Interface simples para construção de fluxos "comuns", desde que não exigem funcionalidades externas.
- **Escalabilidade Horizontal**: *Deploy* de um Cluster de servidores do *Nifi* geridos pelo *ZooKeeper*.
- **Variedade das Fontes de Dados**: Variedade razoável de *Processors* para ingestão de dados.
  - Exemplos: Sistemas de Arquivo, Banco de Dados, APIs e Protocolos de Transferência, E-mail, Serviços de Streaming como Kafka.
- **Processamento em Tempo Real**: Event-driven com baixa latência e alta performance.

5.3 Vantagens

- **Data Lineage Completo**: Rastreabilidade total dos dados para compliance e auditoria.
- **Tolerância a Falhas**: Recuperação automática e gerenciamento de erros robusto.
- **Governança de Dados**: Controle granular de acesso, segurança e políticas de dados.

5.4 Quando Usar NiFi

✔ Ideal para:

Fluxos complexos e dinâmicos

Rastreabilidade completa

Processamento em tempo real

Integração de múltiplas fontes

Equipes não-técnicas

Workflows *event-driven*

✖ Não recomendado para:

Transformações complexas

Processamento batch pesado

Computação intensiva

Modelagem com DAGs é preferível

5.5 Tchau

Figura 2: *Niagara falls.*