HOCHSCHULE FULDA
UNIVERSITY OF APPLIED SCIENCES

AI5101 - Masterseminar: BigData Apache NiFi

Hausarbeit am Fachbereich AI der Hochschule Fulda

Prüfer: Prof. Dr. Christoph Scheich

Autoren:
Nick Stolbov (1269907)
Lukas Gerstacker (1316843)

Eingereicht am 15.01.2025

Inhaltsverzeichnis

Abbildungsverzeichnis

Code-Snippet-Verzeichnis

1 Einleitung

Apache NiFi ist eine leistungsstarke, skalierbare und benutzerfreundliche Plattform für die Automatisierung und Verwaltung von Datenflüssen. Ursprünglich von der NSA entwickelt und später als Open-Source-Software der Apache Foundation übergeben, hat sich NiFi zu einem wichtigen Werkzeug in der modernen Datenverarbeitung entwickelt. Mit seiner grafischen Benutzeroberfläche und einer Vielzahl an integrierten Prozessoren bietet NiFi eine flexible Möglichkeit, Daten aus unterschiedlichen Quellen zu erfassen, zu verarbeiten und an verschiedene Zielsysteme weiterzuleiten. Es ermöglicht dabei sowohl einfache als auch hochkomplexe Datenflüsse. [1]

Diese Arbeit konzentriert sich auf Apache NiFi in der Version 1.0, wobei die meisten der behandelten Konzepte auch auf die aktuelle Version 2.0 übertragbar sind. Ziel ist es, einen praktischen Einstieg in die Installation und Konfiguration zu geben und anhand von Übungen die Anwendung in realistischen Szenarien zu veranschaulichen.

Alle benötigten Dateien für die Installation und die Übungen sind in folgendem GitHub-Repository unter der MIT-Lizenz veröffentlicht:

- https://github.com/Niktron/nifi-project[7]

2 Installation

In diesem Kapitel werden verschiedene Methoden zur Installation und Bereitstellung von Apache NiFi vorgestellt. Zunächst wird eine Bare-Metal Installation vorgestellt, gefolgt von einer detaillierten Betrachtung der Docker-basierten Bereitstellung. Es wird empfohlen, die Bereitstellung über Docker durchzuführen. Einer der Gründe dafür ist, dass es in der Übung später vorausgesetzt wird, weshalb darauf auch mehr im Detail eingegangen wird.

2.1 Bare-Metal

Der Abschnitt konzentriert sich auf die Einrichtung einer einzelnen Apache NiFi-Instanz direkt auf einem Windows-System und dient in diesem Kapitel lediglich als exemplarische Darstellung für eine Bare-Metal Installation. Ziel ist es, die grundlegenden Schritte und Anforderungen einer manuellen Installation zu veranschaulichen. Für die Bereitstellung von Apache NiFi wird jedoch der Docker-Ansatz empfohlen, da dieser nicht nur höhere Flexibilität, Skalierbarkeit und Wartungsfreundlichkeit bietet, sondern auch die Einrichtung und Verwaltung von Clustern deutlich vereinfacht (siehe 2.2).

2.1.1 Systemvoraussetzungen

Apache NiFi kann grundsätzlich auf nahezu jedem handelsüblichen Computer, aber auch innerhalb eines Clusters ausgeführt werden. Die konkret benötigte Hardware ist stark von der Komplexität der verwendeten Dataflows abhängig. Hinsichtlich der Software gelten die folgenden Voraussetzungen: [2]

- **Benötigte Java-Version:** Java 8 oder 11 (NiFi 2.0: Java 21 [3])
- **Unterstützte Betriebssysteme:**
  - Linux
  - Unix
  - Windows
  - macOS
- **Unterstützte Webbrowser:**
  - Microsoft Edge
  - Mozilla Firefox
  - Google Chrome
  - Safari

2.1.2 Download und Startvorgang

Zunächst müssen die Binary Files (NiFi Standard) unter https://nifi.apache.org/download/ heruntergeladen und entpackt werden, siehe Abbildung 1. Anschließend muss im Unterordner `/bin` die Datei `nifi.cmd` (Windows) bzw. `nifi.sh` (Linux) mit dem Parameter `start` geöffnet werden. Zum Stoppen kann der Parameter `stop` und zum Prüfen des aktuellen Status der Parameter `status` verwendet werden. Nach dem erfolgreichen Start kann die Benutzeroberfläche unter `https://localhost:8443/nifi` aufgerufen werden.

Bei dem ersten Start wird ein Standardbenutzer angelegt. Die dazugehörigen Logindaten werden in der Datei `/logs/nifi-app.log` ausgegeben. Eine Änderung ist mit folgendem Befehl möglich: `/bin/nifi.sh set-single-user-credentials<username><password>`

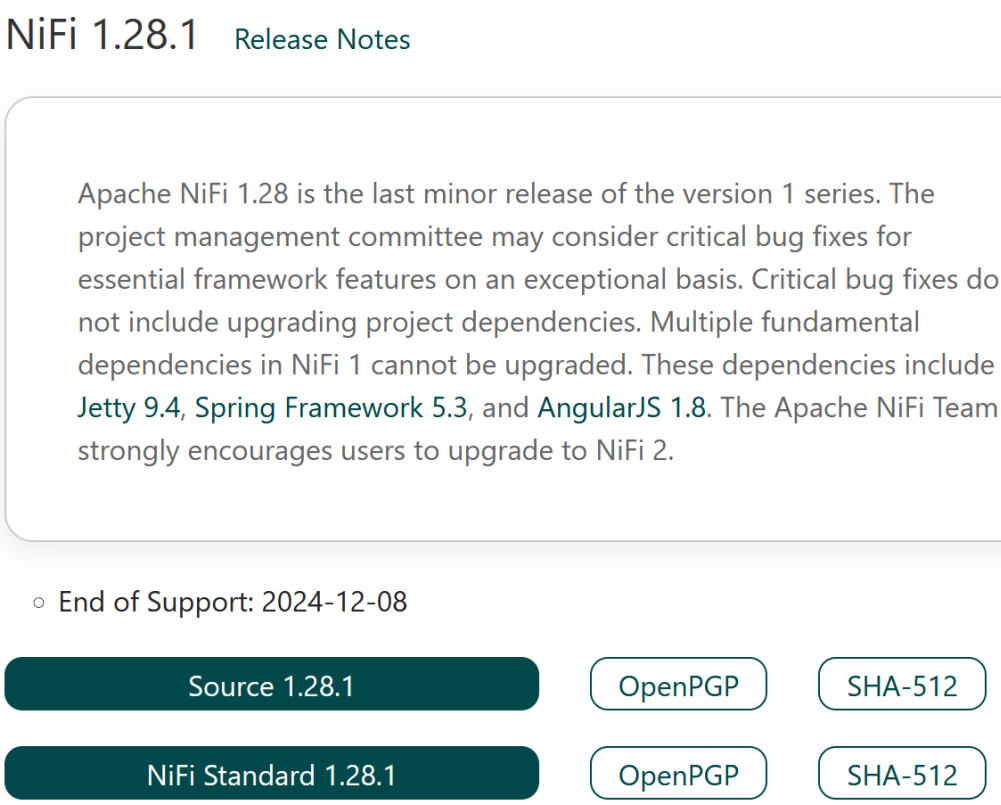

Abbildung 1: NiFi 1.28.1 Download

2.1.3 Konfiguration

Die Konfiguration von NiFi erfolgt hauptsächlich in der Datei `nifi.properties`. Dort kann unter anderem der Standardport für die Benutzeroberfläche angepasst werden.

2.2 Docker

Dieser Abschnitt erläutert die Verwendung von Docker für die Bereitstellung und Verwaltung von einem Apache NiFi Cluster. Docker bietet eine flexible und isolierte Umgebung, die die Installation und Verwaltung vereinfacht, unabhängig vom zugrunde liegenden Betriebssystem. In diesem Kapitel wird dabei speziell auf Docker Compose eingegangen, das die Konfiguration und Verwaltung von Multi-Container-Umgebungen erleichtert. Diese Funktionalität macht es besonders geeignet für die Einrichtung eines Clusters, wie er im weiteren Verlauf des Kapitels vorgestellt wird.

Zunächst werden die Systemvoraussetzungen für eine Docker Bereitstellung erläutert. Daraufhin werden die drei verwendeten Komponenten – Nginx, Apache ZooKeeper und der Apache NiFi Cluster – kurz aufgezeigt. Anschließend wird gezeigt, wie die Bereitstellung konfiguriert wird. Abschließend wird erläutert, wie der Cluster mithilfe dieser Konfiguration verwendet werden kann.

2.2.1 Systemvoraussetzungen

Um Docker nutzen zu können, wird eine Docker Runtime benötigt. Die Installation kann über Docker Desktop (https://docs.docker.com/get-started/get-docker/) erfolgen, das die folgenden Betriebssysteme unterstützt:

- macOS
- Windows
- Linux

Für Linux-Systeme steht als Alternative die Docker Engine (https://docs.docker.com/engine/install/) zur Verfügung. Diese Option ist ressourcenschonender als Docker Desktop und daher besonders für Server-Umgebungen geeignet.

Zusätzlich zu Docker wird Docker Compose benötigt. Die Installation ist abhängig davon, für welche Art der Docker Installation sich im vorherigen Schritt entschieden wurde.

- **Docker Desktop:** Docker Compose ist bereits in Docker Desktop enthalten und erfordert keine separate Installation.
- **Docker Engine:** Bei Verwendung der Docker Engine muss Docker Compose separat installiert werden. Eine detaillierte Installationsanleitung ist in der offiziellen Dokumentation unter https://docs.docker.com/compose/install/ verfügbar.

2.2.2 Komponenten

Für den Betrieb eines NiFi Clusters werden zusätzliche Services benötigt. Im Folgenden werden alle verwendeten Software-Produkte kurz erläutert und was deren Funktion in der Bereitstellung sind erklärt.

Nginx [6]: Nginx ist ein HTTP-Webserver, der in dieser Umgebung als Reverse Proxy eingesetzt wird. Dadurch wird der Zugriff auf den Cluster ermöglicht, ohne dass die Ports der Nodes direkt auf dem Hostsystem freigegeben werden müssen. Dies reduziert die Anzahl der genutzten Ports, da nur der Nginx-Service als einziger Container Host-Ports verwendet. Im Code-Snippet 1 ist die Servicebeschreibung des Nginx-Containers dargestellt.

Code-Snippet 1: Docker Compose - Nginx

```
nginx:
    image: nginx:1.27.2
    restart: always
    env_file: ".env"
    ports:
      - "80:80"
      - "${NIFI_HTTP_PORT}:${NIFI_HTTP_PORT}"
      - "${NIFI_FLOW_HTTP_PORT}:${NIFI_FLOW_HTTP_PORT}"
    networks:
      - nifi_cluster
    configs:
      - source: nginx_conf
        target: /etc/nginx/conf.d/default.conf
    depends_on:
      - nifi
```

Die Servicebeschreibung verwendet die Nginx-Version 1.27.2 (neuste Version, Stand der Erstellung) und enthält spezifische Konfigurationen. Dabei werden die Ports freigegeben, darunter der Standard-HTTP-Port (80) für den Browserzugriff, der interne HTTP-Port der NiFi-Container (als Workaround, da NiFi diesen Port statisch in Redirect-Antworten verwendet) sowie der Flow-Port, der für den eigenen Webserver genutzt wird und für die spätere Übung relevant ist. Die Standardkonfiguration von Nginx wird durch eine benutzerdefinierte Datei (`default.conf`) ersetzt, die die spezifischen Anforderungen des Projekts umsetzt.

Apache ZooKeeper [4]: Apache ZooKeeper ist ein Service-Koordinator, der für den Austausch von Informationen zwischen externen Services eingesetzt wird. Im Kontext des NiFi-Clusters ermöglicht er die Kommunikation zwischen den einzelnen Nodes untereinander und ist für die Bestimmung des Cluster Coordinators sowie der Primary Node verantwortlich. Die Servicebeschreibung ist im Code-Snippet 2 dargestellt.

Code-Snippet 2: Docker Compose - ZooKeeper

```
zookeeper:
    image: zookeeper:3.9.3
    restart: always
    networks:
      - nifi_cluster
    env_file: ".env"
    environment:
      - ALLOW_ANONYMOUS_LOGIN=yes
      - ZOO_MY_ID=1
      - ZOO_SERVERS=server.1=zookeeper:2888:3888;2181
```

Die Servicebeschreibung verwendet ZooKeeper in der Version 3.9.3 (neuste Version, Stand der Erstellung) und enthält spezifische Konfigurationen. Die Authentifizierung ist deakti-

viert, und die Instanz wird als einzelne Node in einem Cluster eingerichtet. Dazu wird die ID auf 1 gesetzt, und die Serverliste enthält lediglich die eigene Adressen-Beschreibung.

Apache NiFi Cluster [1]: Der Apache NiFi Cluster bildet den Kern des Deployments und stellt alle Funktionen von NiFi innerhalb eines redundanten Clusters bereit. Mit den bereitgestellten Containern können Data Flows in NiFi erstellt, bearbeitet, importiert, exportiert und genutzt werden. Die Konfiguration des Clusters ist im Code-Snippet 3 dargestellt.

Code-Snippet 3: Docker Compose - NiFi Cluster

```
nifi:
    image: apache/nifi:1.28.1
    restart: always
    deploy:
      mode: replicated
      replicas: ${NIFI_NODE_COUNT}
      endpoint_mode: vip
      restart_policy:
        condition: on-failure
        delay: 5s
        max_attempts: 3
        window: 120s
    networks:
      - nifi_cluster
    volumes:
      - ./data:/data
    env_file: ".env"
    environment:
      - NIFI_WEB_HTTP_PORT=${NIFI_HTTP_PORT}
      - NIFI_CLUSTER_IS_NODE=true
      - NIFI_CLUSTER_NODE_PROTOCOL_PORT=8082
      - NIFI_ZK_CONNECT_STRING=zookeeper:2181
      - NIFI_ZK_CLIENT_SECURE=false
      - NIFI_ELECTION_MAX_WAIT=20 sec
      - NIFI_SENSITIVE_PROPS_KEY=${NIFI_SENSITIVE_KEY}
    depends_on:
      zookeeper:
        condition: service_started
```

Die Servicebeschreibung verwendet die NiFi-Version 1.28.1 (neuste 1.X Version, Stand der Erstellung) und spezifische Einstellungen. Die einzelnen Nodes werden mithilfe der Docker-Compose-`deploy`-Funktion [5] redundant bereitgestellt. Über eine Umgebungsvariable kann die Anzahl der Nodes gesteuert werden. Der `vip` Endpunkt-Modus erzeugt virtuelle IP-Adressen, sodass der Cluster von außen über eine einzige Domain erreichbar ist, dadurch ist die genaue Anzahl der Nodes erst nach der internen Kommunikation und dem Verbindungsaufbau zwischen den Instanzen ersichtlich.

Für den Datenimport und -export wird ein lokaler Ordner als Volume gemountet, wodurch Daten persistiert werden und über das Dateisystem des Host-OS zugänglich sind. Zusätzlich werden die Nodes so konfiguriert, dass sie die definierten Ports nutzen, im Cluster-Modus arbeiten, den ZooKeeper-Service erreichen können, einen Timeout für die Flowsynchronisation haben und mit einem geheimen Schlüssel für sensible Einstellungen ausgestattet sind.

2.2.3 Konfiguration

Die Konfiguration des Clusters erfolgt durch die Anpassung der Environment-Datei (.env). Alle anderen Anpassungen, die nicht über diese Datei erfolgen können, müssen direkt in der `docker-compose.yaml` durchgeführt werden. Allerdings können die für die Bereitstellung und Verwendung benötigten Konfigurationen vollständig über die Environment-Datei vorgenommen werden. Im unteren Code-Snippet 4 sind alle Einstellungen sichtbar.

Code-Snippet 4: Docker Compose - Environment Datei

```
1 NIFI_HTTP_PORT=8080
2 NIFI_FLOW_HTTP_PORT=8081
3 NIFI_NODE_COUNT=3
4 NIFI_SENSITIVE_KEY=supersecretrandombutfixedstring
```

- **NIFI_HTTP_PORT**: Gibt den HTTP-Port an, durch den der Zugriff auf die NiFi-UI über den Nginx erfolgt. Standardwert ist `8080`.
- **NIFI_FLOW_HTTP_PORT**: Bestimmt den HTTP-Port, der Verwenden werden kann, um weitere Web-Server auf dem Cluster zu erstellen. Dies wird später in der Übung verwendet. Standardwert ist `8081`.
- **NIFI_NODE_COUNT**: Legt die Anzahl der Knoten im NiFi-Cluster fest. Hier wird `3` als Standardwert genutzt.
- **NIFI_SENSITIVE_KEY**: Ein geheimer Schlüssel, der auf allen Knoten identisch sein muss. Dieser Wert sollte einzigartig und sicher gewählt werden.

2.2.4 Bereitstellung

Für die Bereitstellung muss lediglich der Command `docker compose up -d` im Verzeichnis, in dem sich die Datei `docker-compose.yaml` befindet, ausgeführt werden. Daraufhin werden alle benötigten Abhängigkeiten heruntergeladen und gestartet. Hier muss darauf geachtet werden, dass die Docker Runtime auch gestartet ist, ansonsten wird eine Fehlermeldung angezeigt. Das Herunterfahren und Entfernen der Umgebung mit den Komponenten erfolgt dann durch den Befehl `docker compose down`.

Nach dem Start kann der Cluster im Browser über http://localhost erreicht und verwendet werden. Der Cluster braucht einige Zeit für den Start und den Austausch zwischen den Knoten, hier kann es einige Minuten dauern, bis dieser erreichbar ist.

3 Übungen

Mit den folgenden Übungsbeispielen sollen die Grundkonzepte von Apache NiFi vermittelt werden. Das erste Beispiel "Move Files" bietet einen einfachen Einstieg und zeigt grundlegende Funktionen. Der "Department Router" ist eine Erweiterung dieses Beispiels, der ein komplexeres und tieferes Verständnis durch zusätzliche Anforderungen schaffen soll.

Vor der Durchführung muss die NiFi-Installation über Docker, wie in Kapitel 2.2 zuvor erläutert, abgeschlossen und die Benutzeroberfläche geöffnet sein.

3.1 Move Files

In dieser Übung werden Dateien aus dem Eingangsverzeichnis `/data/input` in das Ausgangsverzeichnis `/data/output` verschoben.

Die Lösung dieser Übung ist hier zu finden: GitHub Move Files Template

1. **Vorbereitung der Verzeichnisse**
   - Erstellung des Verzeichnisses `/data/input` im Root-Verzeichnis
   - Erstellung des Verzeichnisses `/data/output` im Root-Verzeichnis
2. **Erstellung des NiFi-Workflows**

   a) **Hinzufügen des `GetFile`-Prozessors**
   - Platzierung des `GetFile`-Prozessors im Workflow.
   - Konfiguration des Prozessors:
     - **Input Directory**: `/data/input`
     - **Keep Source File**: `false` (Dateien werden nach dem Abruf verschoben)
   - Hinweis: Der Prozessor verfügt über keine `failure`-Beziehung. Falls keine Dateien im Verzeichnis vorliegen, wird keine Aktion ausgeführt.

   b) **Hinzufügen des `PutFile`-Prozessors**
   - Platzierung des `PutFile`-Prozessors im Workflow
   - Konfiguration des Prozessors:
     - **Directory**: `/data/output`
   - Verknüpfung des Outputs des `GetFile`-Prozessors mit dem Input des `PutFile`-Prozessors
   - Terminierung aller Beziehungen:
     - **success**: Der Flow wird nach erfolgreichem Speichern beendet.
     - **failure**: Der Flow wird bei Fehlern beendet (z. B. wenn das Zielverzeichnis nicht existiert).

3. **Ausführen des Workflows**

   - Aktivierung beider Prozessoren.
   - Erstellen einer Date im Inputverzeichnis.
   - Überprüfung des Inhalts von `/data/output`, um sicherzustellen, dass die Dateien korrekt verschoben wurden.
   - Kontrolle, dass `/data/input` nach Abschluss des Flows leer ist.

3.2 Department Router

In dieser Übung wird die Implementierung eines erweiterten Apache NiFi-Workflows durchgeführt, der auf HTTP-Requests reagiert. Die eingehenden Daten werden dabei validiert und basierend auf spezifischen Attributen in verschiedene Verzeichnisse routet, während Fehlerfälle effektiv gehandhabt werden. Die notwendigen Verbindungen zwischen den Prozessoren sind der Übersichtlichkeit halber nicht im Detail beschrieben, lassen sich jedoch anhand der Prozessoranordnung und der Workflow-Logik leicht nachvollziehen. Alternativ ist auch das Nachbauen anhand des Templates im GitHub Repository möglich.

Die Lösung dieser Übung ist hier zu finden: GitHub Department Router Template

1. **HTTP Server**

   - **HandleHttpRequest**
     - **Listening Port**: `8081`
     - **HTTP Context Map**: `StandardHttpContextMap` (direkt aktivieren)
     - **Allowed Paths**: `/content`
   - **HandleHttpResponse** für die HTTP-Antwort:
     - **HTTP Status Code**: `200`
     - **HTTP Context Map**: `StandardHttpContextMap`
     - **Relationships**:
       * **failure**: `terminate`
       * **success**: `terminate`

2. **Logik für Department Router**

   a) **Process Group einrichten**

   - Um den komplexen Flow zu kapseln, wird eine **Process Group** genutzt.
   - Kommunikation mit der Process Group erfolgt über:
     - **Input Port**: `Request`
     - **Output Ports**: `Success`, `Failure`

   b) **JSON-Validierung und Verarbeitung**

- **ValidateJson**: Validierung anhand eines **JSON Schemas**:

```
{
    "$schema": "http://json-schema.org/draft-07/schema#",
    "type": "object",
    "properties": {},
    "required": [
        "id",
        "department"
    ],
    "additionalProperties": true
}
```

- **EvaluateJsonPath**: Extraktion von Werten in FlowFile-Attribute.
  - **Destination**: `flowfile-attribute`
  - **New Properties**:
    - **department**: `$.department`
    - **id**: `$.id`

c) **Verarbeiten und Speichern der Daten**

- **PutFile**:
  - **Directory**: `/data/output/http/${department}`
  - Zugriff auf `department` erfolgt über `FlowFile`-Attribute (`${department}`)
- **Output Ports** für Fehler und erfolgreiche Ausführung:
  - `success`, `failure`
- Nutzung eines **Funnel**, um Verbindungen zusammenzufassen und zukünftige Änderungen zu erleichtern

3. **Fehlerfälle**

- **HandleHttpResponse für 400 Bad Request-Response**
  - **HTTP Status Code**: `400`
  - **HTTP Context Map**: `StandardHttpContextMap`
  - **Relationships**:
    - **failure**: `terminate`
    - **success**: `terminate`
- **ReplaceText** zur Verbesserung der Fehlermeldung:
  - **Search Value**: `.*`
  - **Replacement Value**: `{"error": "Bad Request"}`
  - **Evaluation Mode**: `Entire text`
  - **Relationships**:
    - **failure**: `terminate`

4. **Verbesserung der Datei-Benennung**

   - **UpdateAttribute**:
     - Einbau dieses Prozessors in Process Group nach EvaluateJsonPath
     - **filename**: `${id}-${now():format('yyyy-MM-dd-HH-mm-ss')}.json`

5. **Ausführen des Workflows**

   - Sicherstellen, dass alle Fälle, einschließlich Fehlerfälle, korrekt verarbeitet werden
   - Senden gültiger Requests mit verschiedenen Departments zur Überprüfung der Routing-Logik
   - Testen ungültiger Requests, wie z. B. fehlerhafter Body, fehlende Attribute, falsches Datenformat

4 Literaturverzeichnis

[1] Apache Software Foundation. *Apache NiFi*. URL: https://nifi.apache.org/ (aufgerufen am 29. 12. 2024).

[2] Apache Software Foundation. *Apache NiFi 1.0 Documentation*. URL: https://nifi.apache.org/documentation/v1/ (aufgerufen am 22. 12. 2024).

[3] Apache Software Foundation. *Apache NiFi 2.0 Administration Guide*. URL: https://nifi.apache.org/nifi-docs/administration-guide.html (aufgerufen am 22. 12. 2024).

[4] Apache Software Foundation. *Apache ZooKeeper*. URL: https://zookeeper.apache.org/ (aufgerufen am 29. 12. 2024).

[5] Docker Inc. *Compose Deploy Specification*. URL: https://docs.docker.com/reference/compose-file/deploy/ (aufgerufen am 29. 12. 2024).

[6] Nginx. *Nginx*. URL: https://nginx.org/ (aufgerufen am 29. 12. 2024).

[7] Niktron. *GitHub Repository - nifi-project*. URL: https://github.com/Niktron/nifi-project (aufgerufen am 09. 01. 2025).