Duale Hochschule Baden-Württemberg Mosbach

Seminararbeit

Speicherung von MES-Daten (MPDV-Hydra) der "Digitalen Fabrik" in dem HDP (Hortonworks Data Platform)-Cluster

Verfasser:

Christian Bonfert

Katharina Ehrmann

Maria Fladung

Lukas Habermann

Roman Nolde

Tom Schmelzer

Studiengang Wirtschaftsinformatik

Profil: Industrie

Bearbeitungszeitraum: 20. April 2020 – 28. Juni 2020

Kurs:	WI17A
Studiengangsleiter:	Prof. Dr. Dirk Palleduhn
Wissenschaftlicher Betreuer:	Sven Langenecker

Zusammenfassung

Diese Seminararbeit behandelt das Thema der Ablage von Daten auf einem Hortonworks Data Platform-Cluster. Es werden Daten aus der REST API des ME-Systems Hydra der Digitalen Fabrik der DHBW Mosbach ausgelesen und diese auf dem HDP-Cluster der DHBW Mosbach abgespeichert.

Es erfolgt die Erstellung von Konzepten zur Datenablage in HDFS, Apache Hive und Apache HBase und die Prüfung der Eignung der Konzepte für die abzulegenden Daten.

Zusätzlich wird für geeignete Ablagearten ein Performancetest anhand eines repräsentativen API-Abrufs durchgeführt.

Abstract

This seminar paper deals with the topic of storing data on a Hortonworks Data Platform cluster. Data is read from the REST API of the ME-System Hydra of the Digital Factory of the DHBW Mosbach and stored on the HDP-Cluster of the DHBW Mosbach.

Concepts for data storage in HDFS, Apache Hive and Apache HBase are elaborated and the suitability of the concepts for the data to be stored is checked.

In addition, a performance test is conducted for suitable storage types using a representative API call.

Inhaltsverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface
BDE	Betriebsdatenerfassung
CAD	Computer Aided Design
DHBW	Duale Hochschule Baden-Württemberg
ERP	Enterprise Ressource Planning
GUI	Graphical User Interface
HATEOAS	Hypermedia as the Engine of Application State
HDFS	Hadoop Distributed File System
HDP	Hortonworks Data Platform
HQL	Hive Query Language
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JDBC	Java Database Connectivity
JSON	JavaScript Object Notation
MES	Manufacturing Execution System
PPS	Produktionsplanung und -steuerungssystem
PyPI	Python Package Index
RAM	Random Access Memory
REST	Representational State Transfer
SASL	Simple Authentication and Security Layer
SQL	Structured Query Language
UMCM	Universal Machine Connectivity for MES
URL	Uniform Resource Locator
VDI	Verein Deutscher Ingenieure
XML	Extensible Markup Language
YARN	Yet Another Resource Negotiator

Abbildungsverzeichnis

1 Einleitung

Im Zeitalter von Industrie 4.0 und Big Data spielen große Datenmengen eine entscheidende Rolle. Diese müssen jedoch erst gesammelt und abspeichert werden. Zur Abspeicherung bietet sich eine verteilte Speicherung an, da diese gut mit den wachsenden Datenmengen und den darauf aufbauenden Anforderungen der Echtzeitanalyse skaliert, wie sie beispielsweise in der Hadoop-Distribution HDP (Hortonworks Data Platform) verwendet wird. (vgl. [Posey (2013)])

Auch die DHBW (Duale Hochschule Baden-Württemberg) Mosbach nutzt ein HDP-Cluster zur Speicherung großer Datenmengen. Dieses soll nun um die Speicherung von MES (Manufacturing Execution System) Daten aus der Digitalen Fabrik der DHBW Mosbach erweitert werden. Die Daten werden über eine REST (Representational State Transfer) API (Application Programming Interface) des ME-Systems Hydra der Firma MPDV Microlab GmbH bereitgestellt, das in der Digitalen Fabrik eingesetzt wird.

Ziel der Arbeit

Das Ziel dieser Seminararbeit ist es, mögliche Datenablagearten im HDP-Cluster zu eruieren und eine für die bereitgestellten Daten sinnvolle Lösung zu entwickeln. Die Daten sollen fortlaufend in bestimmten Intervallen abgespeichert werden können, um eine Historie der über die API bereitgestellten Daten zu erhalten.

Aufbau der Arbeit

Zu Beginn der Arbeit werden einige theoretische Grundlagen gelegt.

Diesen folgt eine Vorstellung der Digitalen Fabrik und die Sichtung und Untersuchung der MES Hydra REST API.

Anschließend werden die möglichen Datenablagearten im HDP-Cluster der DHBW-Mosbach aufgezeigt. Für diese erfolgt die Konzeption und Umsetzung der Speicherung der API-Daten mittels der Verwendung einer auf dem HDP-Cluster installierten Software (Apache NiFi) sowie einer Eigenentwicklung (Programmiersprache Python).

Dem daran anknüpfenden Performancetest der geeigneten Lösungen folgen ein abschließendes Fazit und Anmerkungen zur Automatisierung und Weiterentwicklungsmöglichkeiten der möglichen Lösung.

2 Theoretische Grundlagen

Dieser Abschnitt soll eine theoretische Übersicht über die für diese Arbeit wichtigen Technologien und Systeme bieten. Auf diesem Wissen baut die praktische Ausarbeitung auf.

2.1 REST API

Eine REST API ist eine Programmierschnittstelle bzw. Anwendungsschnittstelle, welche die Eigenschaften des REST-Programmierparadigmas erfüllt.

Ein Dienst ist REST-konform sofern er folgende sechs Eigenschaften erfüllt:

1. **Client-Server:** Es müssen alle Eigenschaften einer Client-Server Architektur gegeben sein. Ein Client kann also einen Dienst aufrufen, welcher von einem Server bereitgestellt wird.
2. **Zustandslosigkeit:** Die Kommunikation zwischen Client und Server muss zustandslos (stateless) sein. Das heißt eine Anfrage an den Server muss sämtliche Informationen enthalten damit dieser sie verstehen/verarbeiten kann. Der Server speichert also keine Zustände. Somit werden Zustände ausschließlich auf der Clientseite gespeichert.
3. **Caching:** Durch Caching wird die Netzwerkauslastung verringert. Hat ein Client bereits eine Antwort auf eine Anfrage erhalten so kann er diese zwischenspeichern (cachen) und für eine äquivalente Anfrage verwenden.
4. **Einheitliche Schnittstelle:** Die Einheitlichkeit einer Schnittstelle wird durch vier weitere Eigenschaften gesichert:
    - **Adressierbarkeit von Ressourcen:** Jede Ressource ist über einen Ressourcen Identifier (RI) adressierbar. Da die REST-Architektur lediglich ein Paradigma ist, liegt es an der konkreten Implementierung einen geeigneten Ressourcen Identifier zu verwenden (z. B. URL (Uniform Resource Locator)).
    - **Repräsentationen zur Veränderung von Ressourcen:** Ressourcen können verschiedene Repräsentationen haben. Je nachdem was angefordert wird kann ein REST-konformer Server die Ressource in der angeforderten Repräsentation ausliefern (beispielsweise in verschiedenen Sprachen oder Dateiformaten).
    - **Selbstbeschreibende Nachrichten:** Durch die Verwendung von Standartmethoden wird eine Selbstbeschreibung erreicht. Die konkrete Implementierung von Standardmethoden über HTTP (Hypertext Transfer Protocol) ist wie folgt:
        - GET: Fordert die Ressource an und ändert nicht den Zustand am Server.

    - POST: Fügt eine neue Ressource hinzu und kann auch weitere Operationen abbilden.
    - PUT: Angegebene Ressource wird angelegt. Falls sie bereits existiert wird diese verändert.
    - DELETE: Löscht die angegebene Ressource.
- **Hypermedia as the Engine of Application State (HATEOAS):** Bei diesem Entwurfsprinzip wird eine REST-Schnittstelle ausschließlich durch und über den Server bereitgestellte URLS navigiert. Die Bereitstellung der RI erfolgt hierbei über Hypermedia z. B. in Form von "href oder "src"-Attributen bei HTML (Hypertext Markup Language) oder bei JSON (Java Script Object Notation) oder XML (Extensible Markup Language) über festgelegte Attribute oder Elemente.

5. **Mehrschichtige Systeme:** Bei mehrschichtigen Systemen kann jede Komponente nur mit der mit ihr direkt in Verbindung stehenden Komponente kommunizieren.
6. **Code on Demand (Optional):** Code zur lokalen Ausführung kann an den Client übertragen werden. Beispielsweise als Java-Script.

(vgl. [Fielding (2000)])

Im Zuge dieser Arbeit ist mit REST API immer die implementierte MES Hydra REST API der digitalen Fabrik der DHBW Mosbach gemeint. Über diese Schnittstelle lassen sich MES-Daten in Form von JSON-Dateien abrufen. Zur Kommunikation mit der Schnittstelle wird das Hypertext Transfer Protocol (HTTP) verwendet. Für Anfragen an die API wird auf die HTTP-Methode GET zurückgegriffen.

2.2 Manufacturing Execution System

Möglichst optimale Produktionsabläufe sind heute für Fertigungsunternehmen ein wichtiges Mittel zur Sicherung der Wettbewerbsfähigkeit auf dem lokalen und globalen Markt. Fehlmengen in der Produktion sollen frühzeitig erkannt oder im besten Fall noch vor der Entstehung vermieden werden, um die daraus resultierenden Kosten auf ein Minimum zu senken. Dazu ist es nötig einen vollständigen Überblick über die einzelnen Schritte der Produktion, sowie den Status des (Zwischen-)Produktes selbst zu kennen (vgl. [Kletti et al. (2019)], S. 1ff). In den 80er Jahren des letzten Jahrhunderts kamen so die ersten Ideen zum Manufacturing Execution System (MES) auf. Es dauerte jedoch bis in die Mitte der 90er Jahre, bis das Thema durch einige Hersteller wieder aufgegriffen wurde, um ihre Erfassungssysteme mit anderen angrenzenden Systemen zu verbinden und so einen Mehrwert für den Kunden zu schaffen (z. B. Personalzeiterfassung mit Betriebsdatenerfassungsterminals). (vgl. [Kletti (2015)], S. 19)

Ein MES ist ein prozessnahes Fertigungsmanagementsystem. Im Gegensatz zum ERP-System, das auf der Managementebene angesiedelt ist und alle Geschäftsprozesse abbildet, arbeitet das ME-System auf der Fertigungsleiterebene mit den operativen Prozessen. Seine Daten bezieht es von den Anlagen und

Maschinen auf der Fertigungs- oder Produktionsebene (vgl. [Brauckmann (2019], S. 241). Es schließt somit die Lücke zwischen dem ERP-System und der Anlagensteuerung, auch Production Floor genannt, und bietet dadurch in Echtzeit mehr Transparenz im Wertschöpfungsprozess (vgl. [Gerberich (2011)], S. 37).

Welche Aufgaben das MES unterstützt, wurde durch den Verein Deutscher Ingenieure (VDI) in der Richtlinie 5600 definiert. Diese zehn Bereiche sind Auftragsmanagement, Feinplanung und Steuerung, Betriebsmittelmanagement, Informationsmanagement, Qualitätsmanagement, Materialmanagement, Personalmanagement, Datenerfassung, Leistungsanalyse und Energiemanagement. In der nachfolgenden Abbildung 1 sind diese, sowie das MES selbst, in den Unternehmenskontext eingeordnet dargestellt. (vgl. [Kletti et al. (2019)], S. 2-4)

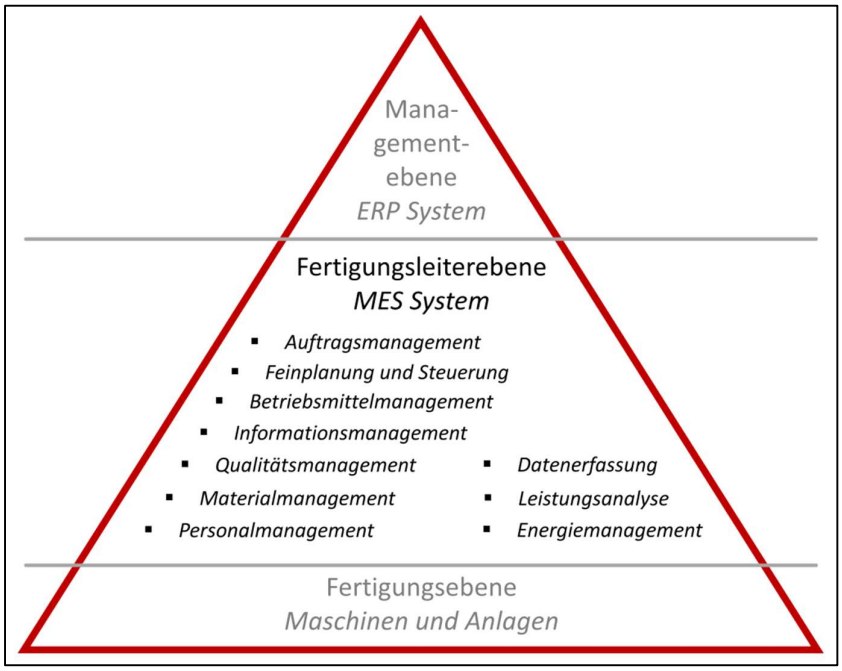


Abbildung 1 MES in Unternehmensebenen
(Eigene Darstellung nach [Kletti et al. (2019)])

Um diese breit gefächerten Aufgaben unterstützen zu können, ist eine umfassende Datenbasis wichtig. Das MES benötigt u. A. umfassende Daten zu Maschinen, Material, Werkzeugen, Personal und Qualität. Außerdem die Auftragsstammdaten über den Artikel, Kunden und die Prüfpläne. Diese Daten kommen traditionell aus vielen verschiedenen Systemen. Die Daten der Maschinen und Anlagen direkt in der Produktion, wie beispielsweise Fräsmaschinen, Waagen, Laboranalysesystemen usw. stehen jedoch im Fokus des MES. Um die Kommunikation zwischen dem System und den verschiedensten Anlagen zu vereinfachen, wurde von der Firma MPDV eine universelle Maschinenschnittstelle entwickelt. Der Datenaustausch erfolgt dabei durch einfache Datentelegramme, die durch die Universal Machine Connectivity for MES (UMCM) definiert sind, welche auf der VDI-Richtlinie 5600 basiert. UMCM sieht die Nutzung einer standardisierten Transportschicht (wie z. B. XML) vor und definiert gleichzeitig den logischen Aufbau auf der Anwendungsschicht. In den Nachrichten können z. B. Zählerstände,

Maschinenstatus, Materialdaten oder Prozesswerte zusammen mit den entsprechenden Zeitstempel übertragen werden (vgl. [Brauckmann (2019)], S. 244-247). Diese Schnittstelle ist für die Nutzung offen, es entstehen keine Lizenzgebühren. Jedoch muss ein Markenlizenzvertrag mit der Firma MPDV vor der Nutzung abgeschlossen werden. (vgl. [fraunhofer.de (2015)])

2.3 MPDV Hydra

HYDRA ist ein von der Firma MPDV Mikrolab GmbH mit Sitz in Mosbach entwickeltes Manufacturing Execution System (vgl. [MPDV (o. J.)]). Die Anforderungen der VDI-Richtlinie 5600 werden vollständig abgedeckt und die Lösung ist modular aufgebaut, sodass das MES-System die drei großen Bereiche Fertigung, Personal und Qualität abdeckt. Abbildung 2 stellt die verschiedenen Funktionen von Hydra dar, welches sich in die bestehende IT-Landschaft integriert und somit als Bindeglied zwischen dem Management (ERP-System) und der Fertigung steht.

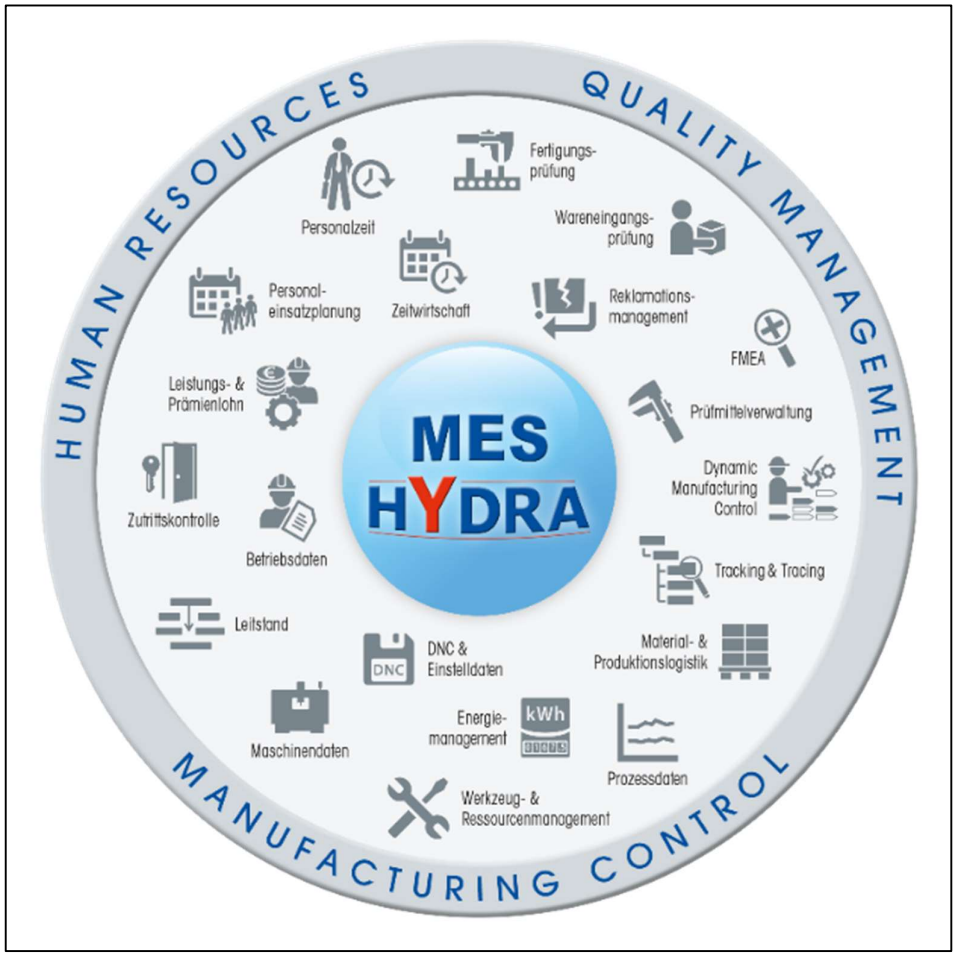


Abbildung 2 MES Hydra
(vgl. [MPDV (o. J.)])

Im Folgenden sollen einige dieser Funktionen genauer erläutert werden.

Betriebsdatenerfassung

Die Betriebsdatenerfassung (BDE) beschreibt allgemein die Datenerhebung im Kontext der Bearbeitung von Fertigungsaufträgen und den dazugehörigen Arbeitsaufträgen. Hierbei sollen beispielsweise eine verbesserte Termintreue und eine kürzere Durchlaufzeit gewährleistet werden.

Maschinendatenerfassung

Ergänzend zu der BDE hilft die Maschinendatenerfassung die zur Verfügung stehenden Maschinen optimal auszunutzen, die Auslastung zu visualisieren und Daten automatisch oder manuell zu erfassen. Die Maschinendatenerfassung erhöht die Transparenz der Maschinenzustände und verbessert den Instanthaltungsprozess.

Werkzeug- und Ressourcen-Management

Zur Verwaltung und Organisation von Werkzeugen sowie Ressourcen, die im Zusammenhang mit der Fertigung stehen, bietet MPDV HYDRA-WRM an. Über den kompletten Lebenszyklus kann die Nutzung von Werkzeug (Verschleiß, etc.) dokumentiert, protokolliert und ausgewertet werden.

Einstelldatenübertragung

Ein entscheidender Wettbewerbsfaktor ist der Automatisierungsgrad in der Fertigung im Unternehmen (vgl. [Kletti (2015)] S. 94). Die automatisierte Übertragung der Einstelldaten von NC-Programmen an die verschiedenen Maschinen gewährt eine höhere Prozesssicherheit und kürzere Rüstzeiten.

Prozessdatenverarbeitung

HYDRA-PDV bietet eine integrierte Sicht über die Herstellungsprozesse aufgrund einer gemeinsamen Datenbasis.

Die digitale Fabrik der DHBW Mosbach fokussiert sich bei den MES-Daten, welche von MPDV HYDRA übertragen werden, auf die Betriebs- und Maschinendaten.

2.4 Hadoop und die Hortonworks Data Platform

Apache Hadoop

Hadoop ist ein, in Java geschriebenes, Open-Source-Software-Framework. Mit Hilfe dieses Frameworks können verteilte, skalierbare Systeme realisiert werden. Es besteht aus den vier Kernkomponenten Hadoop Common, HDFS (Hadoop Distributed File System), Map Reduce Algorithmus und YARN (Yet Another Resource Negotiator) (vgl. [Luber et al. (2016)]).

Hadoop Common

Hadoop Common enthält die grundlegenden Bibliotheken und Werkzeuge die andere Hadoop Module benötigen und stellt diese über Schnittstellen bereit. Beispiele für solche Bibliotheken und Werkzeuge

sind Java-Archiv-Dateien und Skripte, die den Start der Software ermöglichen (vgl. [Luber et al. (2016)]).

HDFS

Bei HDFS handelt es sich um ein Dateisystem, das die verteilte Speicherung von Daten ermöglicht (vgl. [Luber et al. (2016)]).

Map Reduce Algorithmus

Dieser Algorithmus ermöglicht die Aufspaltung von rechenintensiven Aufgaben in kleinere Arbeitspakete, die von mehreren Rechnern parallel verarbeitet werden können. Nach der Abarbeitung der einzelnen Arbeitspakete führt der Algorithmus die Ergebnisse zusammen (vgl. [Luber et al. (2016)]).

YARN

YARN ist ein Ressourcen Manager und eine Erweiterung des Map Reduce Algorithmus. Er dient der Verwaltung von Ressourcen in einem Cluster und kann den Cluster-Komponenten verschiedene Aufgaben dynamisch zuweisen (vgl. [Luber et al. (2016)]).

Hadoop im Business Intelligence-Umfeld

Da das Hadoop-Framework die Verarbeitung von großen Datenmengen ermöglicht, die sowohl strukturiert als auch unstrukturiert sein können, ist es ideal für den Einsatz im Business Intelligence-Umfeld geeignet. Durch die parallele Datenverarbeitung auf verteilten Systemen können schnell hochkomplexe Analysen auf der Basis von riesigen Datenmengen erstellt werden (vgl. [Luber et al. (2016)]).

Unterschiedliche Distributionen

Ähnlich wie beim Betriebssystem Linux, gibt es von Hadoop unterschiedliche Distributionen von verschiedenen Herstellern. Diese bieten neben den Grundfunktionen von Hadoop weitere, distributionsspezifische Module. Anbieter solcher Distributionen sind beispielsweise Cloudera, Hortonworks oder MapR. Auf die Hortonworks-Distribution wird im nächsten Unterkapitel genauer eingegangen.

Hortonworks Data Platform

Die Hortonworks Data Platform (HDP) ist eine quelloffene Distribution von Hadoop, die von der Firma Hortonworks entwickelt wird. Im Grunde ist die HDP eine erweiterte Form des Hadoop Frameworks. Neben den bereits genannten Grundfunktionen, implementiert Hortonworks in seinem System weitere Funktionalitäten wie z. B. die Anbindung an verschiedene Cloud-Dienste wie AWS (Amazon Web Services) oder Microsoft Azure sowie die Möglichkeit der Containerisierung von Anwendungen (vgl. [Tutanch et al. (2017)]).

3 Rahmenbedingungen

Im folgenden Abschnitt wird die aktuelle Situation beschrieben. Hierfür erfolgt eine Darstellung der Digitalen Fabrik der DHBW Mosbach und die Sichtung der MES Hydra REST API.

3.1 Digitale Fabrik

Von der komplett manuellen Planung des Produktionsbereichs über den vereinzelten Einsatz von beispielsweise Computer Aided Design (CAD), Produktionsplanung und -steuerungssystemen (PPS), der Einbindung des Internets und der digitalen Abbildungen von Produkten geht die Entwicklung hin zur Digitalen Fabrik (vgl. [Bracht et al. (2018)], S. 2-8).

In einer Digitalen Fabrik werden digitale Modelle, Methoden und Werkzeuge netzwerkartig zusammengefasst (vgl. [VDI (2008)], S. 3). Ein durchgängiges Datenmanagement der Komponenten soll zur Erreichung eines hohen Grades an Integration führen (vgl. [VDI (2008)], S. 3).

Als Ziel der Digitalen Fabrik nennt der VDI (2008, S. 3) "die ganzheitliche Planung, Evaluierung und laufende Verbesserung aller wesentlichen Strukturen, Prozesse und Ressourcen der realen Fabrik in Verbindung mit dem Produkt." So stellt eine Digitale Fabrik die Basis für Industrie 4.0 und Smart Factory dar (vgl. [Bracht et al. (2018], S. 2, 24-25).

Die Digitale Fabrik der DHBW Mosbach (Kompetenzzentrum "Fertigungs- und Informationsmanagement") bildet modellhaft die Produktion eines Unternehmens ab. Hierbei spielen industrielle Komponenten (z. B. Produktionsanlagen), Automatisierungstechnik und echtzeitfähige Simulationssysteme eine große Rolle. Informationssysteme (z. B. ERP und MES) vernetzen diese Komponenten und dienen zur Analyse und der Steuerung der Digitalen Fabrik. (vgl. [DHBW Mosbach (o. J.)])

Abbildung 3 zeigt den Aufbau, die Verwendung und Vernetzung der verschiedenen Informationssysteme in der Digitalen Fabrik der DHBW Mosbach. Die Integration der einzelnen Systeme und der Datenaustausch sind hier integraler Bestandteil.

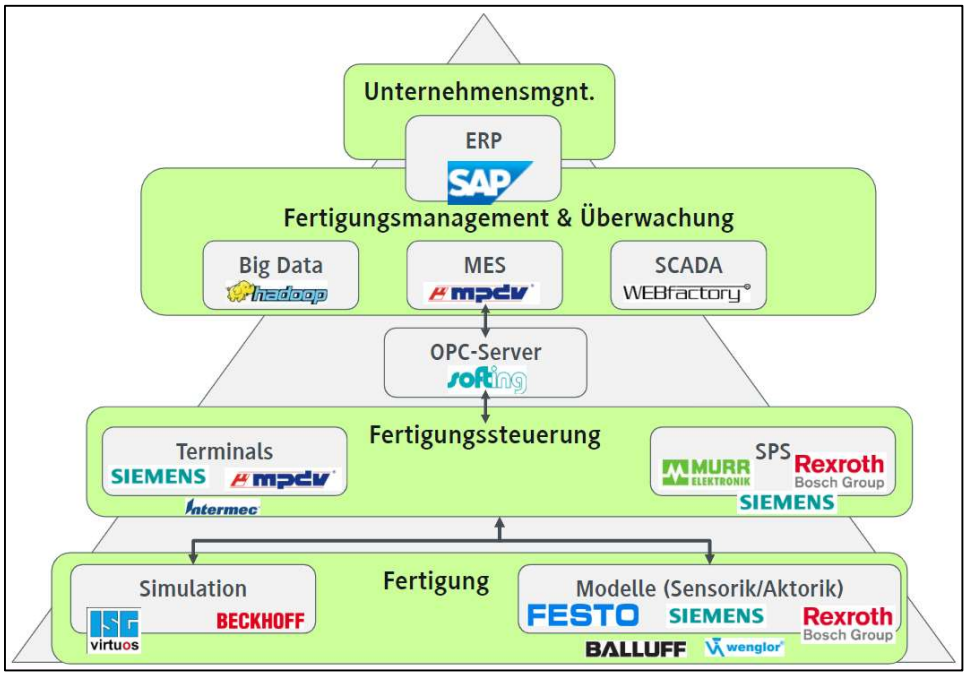


Abbildung 3 Aufbau der Digitalen Fabrik der DHBW Mosbach
(vgl. [Auch et al. (2017)], S. 6)

Daten und Informationen sind ein Hauptbestandteil der Digitalen Fabrik, die an allen Stellen benötigt und auch erzeugt werden. Die Menge an Daten bietet Potenzial zur erweiterten Analyse und Verarbeitung. Als vorbereitender Schritt dazu sollen die Daten aus der Digitalen Fabrik gesichert werden.

Die MES Hydra REST API-Schnittstelle der Digitalen Fabrik der DHBW Mosbach liefert die Daten, welche in dieser Seminararbeit weiterverarbeitet und in einem HDP-Cluster gespeichert werden.

3.2 Sichtung MES Hydra – REST API

Die in der Programmiersprache Java entwickelte MES Hydra REST API der Digitalen Fabrik der DHBW Mosbach ist unter der IP-Adresse *10.50.12.131* und dem Port *8080* (Produktionssystem) zu erreichen. Da nur Daten abgefragt werden sollen, erfolgt in dieser Seminararbeit ausschließlich die Verwendung von GET-Anfragen. Antworten liefert die API im JSON-Format.

JSON ist ein menschenlesbares Datenaustauschformat, das auf der Programmiersprache JavaScript basiert. Zentrale Elemente sind Name-Wert-Paare und Listen, mithilfe derer Daten strukturiert präsentiert werden können. Durch die Verwendung der Name-Wert-Paare lassen sich Objekte darstellen. (vgl. [JSON.org (o. J.)])

Abbildung 4 stellt beispielhaft eine Liste mit zwei Objekten dar. Jedes Objekt enthält ein Name-Wert-Paar, dem als Wert wiederum ein Objekt mit zwei Name-Wert-Paaren zugewiesen ist. Eckige Klammern umschließen eine Liste. Objekte werden von geschweiften Klammern umschlossen.

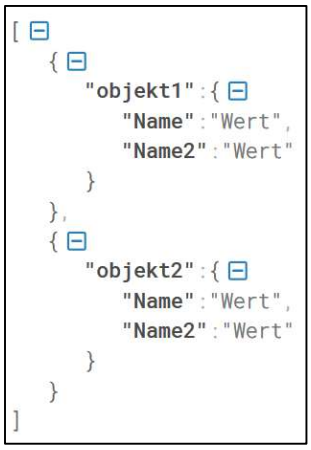


Abbildung 4 JSON Syntax

Die API gliedert sich auf oberster Ebene in zwei Zweige:

- <serverURL>/meta
- <serverURL>/data

Der Endpunkt <server*URL>/meta* enthält Metadaten zur API und lässt Rückschlüsse auf den API-Aufbau zu. Als Antwort wird eine Liste mit 2656 Einträgen geliefert. Abbildung 5 zeigt einen kleinen Ausschnitt. Hierbei handelt es sich um eine Übersicht der vorhandenen Services (vgl. [MPDV Mikrolab GmbH (2018), S. 23]). Die Einträge folgen der Struktur *Domain.Service*, die an Klassen und Methoden aus der Objektorientierung erinnern.

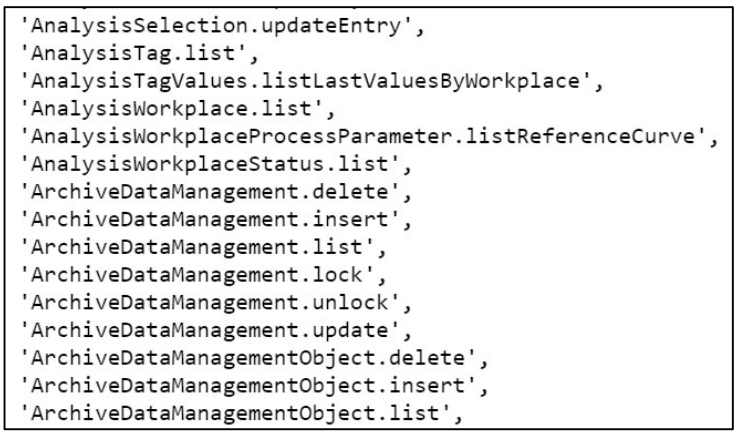


Abbildung 5 Ausschnitt API-Antwort "Meta"

Eine Auswertung mithilfe von Python in einem Jupyter-Notebook (siehe 'API-Inspection' im GitHub-Repository) ergab, dass in der Liste 526 verschiedene Domains und 422 verschiedene Services vorhanden sind, wobei nicht bei jeder Domain jeder Service vertreten ist. Die acht häufigsten vertretenen

Services sind in Abbildung 6 zusammen mit der ermittelten Anzahl gelistet. Alle weiteren sind oft nur bei einer Domain vorhanden.

Service	Anzahl
list	462
update	310
delete	303
lock	303
unlock	303
insert	302
copy	78
new	52

Abbildung 6 Häufigste Services und deren Anzahl

Die Definitionen der einzelnen Services lassen sich über *<serverURL>/meta/<Domain>/<Service>* abrufen (vgl. [MPDV Mikrolab GmbH (2018), S. 24]). Hier ist z. B. angegeben, welche Filter-Parameter bei einer API-Anfrage an den entsprechenden Service mitgegeben werden können.

Der zweite Zweig der API (*<serverURL>/data*) führt zu den tatsächlichen Daten. Hier wird ein Service tatsächlich aufgerufen. Die Endpunkte folgen dem Aufbau *<serverURL>/data/<Domain>/<Service>*. (vgl. [MPDV Mikrolab GmbH (2018), S. 27])

Für diese Seminararbeit sollen Daten aus der API gelesen werden. Bei Services wie *update*, *delete*, *lock*, usw. lässt die Benamung den Rückschluss zu, dass diese zur Manipulation der Daten verwendet werden können. Eine nähere Betrachtung der Endpunkte *<serverURL>/data/<Domain>/list* zeigt, dass diese zum entsprechenden Lesen der Daten geeignet sind. Als Antwort liefert die API jeweils eine Liste mit den Objekten bzw. Datensätzen der entsprechenden Domain.

Betrachtet man nochmals alle verfügbaren Services, zeigt sich, dass einige Domains einen Service haben, dessen erster Teil der Benamung ebenfalls 'list' ist (z. B. 'listByOrder'). Bei der Auswertung aller Domains mit einem Service namens 'list' oder deren Servicename mit 'list' beginnt, ergeben sich 548 Endpunkte, die zur Sammlung aller Daten der API abgerufen werden sollten.

Die meisten "list…"-Services sind ohne Parameter aufrufbar und liefern alle vorhandenen Daten zurück. Einige erwarten jedoch zwingend einen oder mehrere Parameter, die im JSON-Format bei der Anfrage mitgegeben werden müssen. Diese Parameter unterscheiden sich von Domain zu Domain und müssen jeweils manuell ermittelt und deren Inhalt bestimmt werden.

Der Aufbau der Antworten (Standardkonfiguration, keine Übergabe von angepassten Parametern) der *list*-Services wird anhand der Domain BOOrder gezeigt. Wie in Abbildung 7 zu sehen, widmet sich das erste Objekt der JSON-Liste den Metainformationen zur Domain ("__rowType":"META"). Die darauffolgenden "DATA"-Einträge ("__rowType":"DATA") repräsentieren jeweils einen Datensatz bzw. ein Objekt der Domain BOOrder. Bei allen Einträgen ist als Typ 'order' angegeben ("__type":"order") und sie enthalten ein Name-Wert-Paar mit dem Namen 'data'.

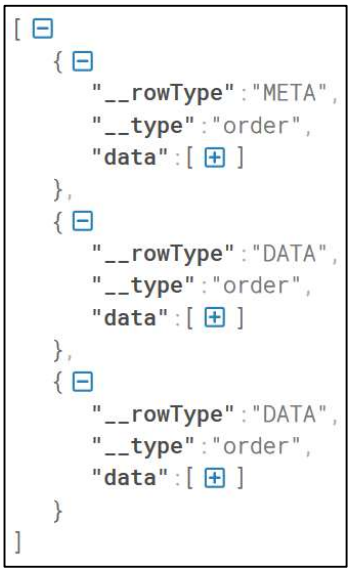


Abbildung 7 Aufbau der *list*-Antwort

Hinter diesem Name-Wert-Paar verbergen sich die eigentlichen Informationen. Der "META"-Eintrag enthält hier eine Liste mit allen Attributen der Domain und deren jeweiligem Datentyp (siehe Abbildung 8 links).

Bei den "DATA"-Einträgen enthält der Wert zum Namen "data" eine Liste mit den Daten pro Attribut (siehe Abbildung 8 rechts). Diese Liste bildet einen Datensatz ab.

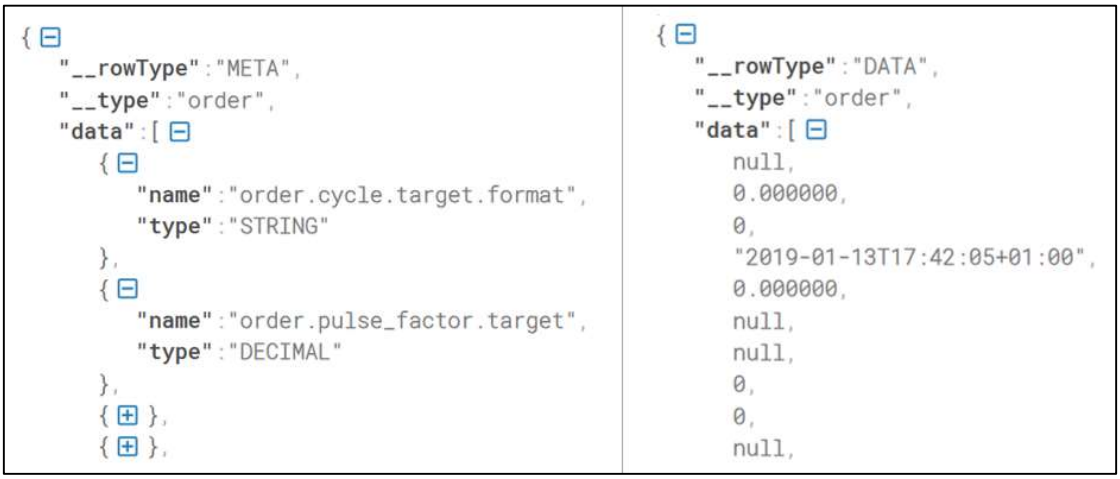


Abbildung 8 Aufbau Meta- und Data-Einträge der *list*-Antwort

Der Aufbau dieser API-Antwort (Standardkonfiguration) erinnert an eine Tabelle. Die erste Zeile enthält die Spaltenbeschriftungen (Metainformationen zum Aufbau der Daten-Einträge) und alle weiteren enthalten die einzelnen Datensätze. Das zeigt, dass die Daten zur Ablage in Tabellen geeignet sind. Generell liefert die API strukturierte Daten, was die Daten auch zur Ablage in Tabellen qualifiziert.

4 Konzeption und Implementierung der Datenablage

Dieses Kapitel beschäftigt sich mit den Arten der Datenspeicherung im HDP-Cluster. Zunächst werden drei mögliche Arten der Datenspeicherung festgelegt. Die Datenspeicherung wird für jede Ablageart mithilfe einer Marktlösung und einer Eigenentwicklung getestet. Als Marktlösung wird auf Apache NiFi zurückgegriffen, die Eigenentwicklung erfolgt mit der Programmiersprache Python. So soll getestet werden, welche Möglichkeit sich für die Daten der MES Hydra API besser eignet.

4.1 Arten der Datenspeicherung im HDP-Cluster

Im HDP-Cluster der DHBW Mosbach sind drei Services zur Datenablage und -speicherung vorhanden. Diese Services sind HDFS, Apache Hive und Apache HBase:

HDFS

Das Hadoop Distributed File System (HDFS) ist ein verteiltes Dateisystem, welches für die Ausführung auf Standardsoftware ausgelegt ist (vgl. [Apache Software Foundation (o. J. a]). Wobei HDFS darauf ausgelegt ist, große Datenmengen (im Gigabyte / Terabyte Bereich) zu verarbeiten.

Hierbei gibt es keine vorgegebene Art des Formats der Datenspeicherung, somit können beispielsweiße JSON-Files, CSV-Dateien aber auch Datenbanken in HDFS hochgeladen und abgerufen werden.

HDFS ist in einer Master/Slave-Architektur aufgebaut. Diese zeichnet sich dadurch aus, dass der Master die Koordination aller Aufgaben übernimmt und diese an die Slaves verteilt. Das Cluster besteht aus einem NameNode (Master) und aus einer Reihe von DataNodes (Slaves). Die Speicherung und Anzeige von Daten ist in Abbildung 9 dargestellt.

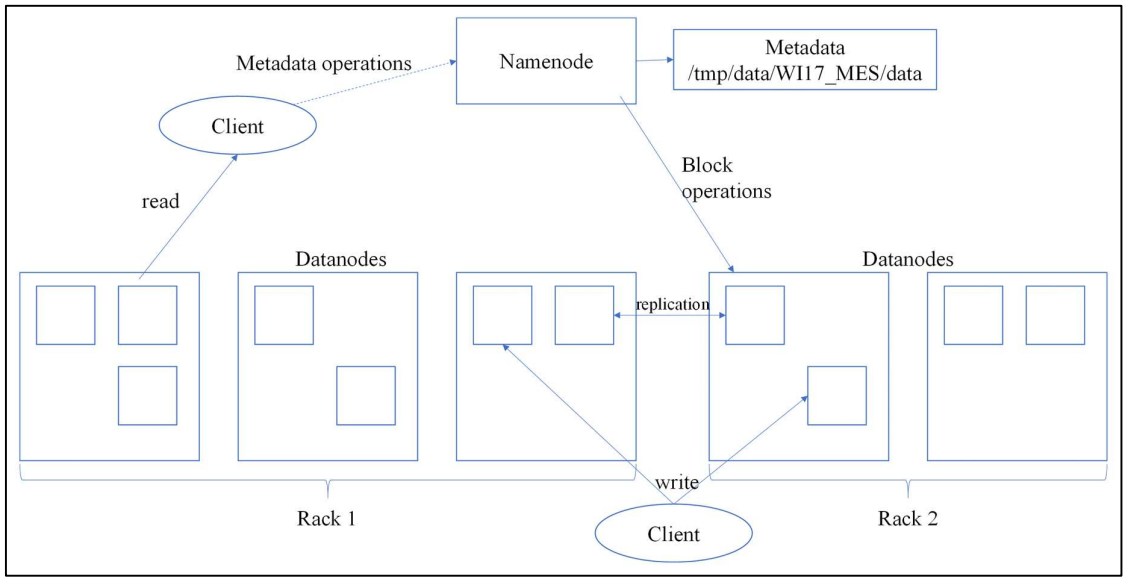


Abbildung 9 Aufbau HDFS

(Eigene Darstellung nach [Apache Software Foundation (o. J. a)])

Im Bereich des HDFS existiert das sog. "Small-files-problem" (vlg [Blaint (2009)]). Es beschreibt ein Problem, welches im HDFS bei der Speicherung kleiner Dateien auftritt (kleiner als die Block-größe). Dabei kann HDFS die Daten nicht mehr effizient speichern, da die Dateien immer in einer vorgeschriebenen Paketgröße gespeichert werden. Sind die zu speichernden Dateien kleiner als diese, nehmen sie dennoch die Größe eines ganzen Paketes ein, somit entsteht nicht nutzbarer/ungenutzter Speicherplatz.

Apache Hive

Apache Hive ist die Open-Source Data Warehouse-Erweiterung für Hadoop von der Apache Software Foundation, welche das Lesen, Schreiben und Verwalten großer Datenmengen, die sich auf verteilten Speichern befinden, erleichtern soll. Es bietet zudem die Möglichkeit, eine Struktur über eine Vielzahl von Datenformaten aufzuerlegen und somit den in HDFS üblicherweise unstrukturierten Daten eine Struktur zu geben. (vgl. [Apache Software Foundation (o. J. b)])

Ermöglicht werden die o. g. Funktionen durch einen SQL(Structured Query Language)-ähnlichen Zugang. Bei diesem handelt es sich um die durch Hive bereitgestellte Abfragesprache Hive Query Language (HQL). HQL hat eine ähnliche Semantik wie SQL-Standards bei relationalen Datenbanken und ist deshalb mit Vorkenntnissen in SQL-Sprachen leicht verständlich. Zudem bietet HQL den Vorteil, diese auch direkt auf MapReduce, Tez und Spark (Programme, die auf Hadoop basieren) anzuwenden, um dort die Performance zu steigern.

Das Datenmodell von Hive wird als hochrangige, tabellenähnliche Struktur über dem HDFS bezeichnet und unterstützt dabei grundlegend drei verschiedene Datenstrukturen: Tabellen, Partitionen und Buckets, die theoretisch in gleicher Reihenfolge immer weiter aufgeteilt werden können. (vgl. [Du (2018)], S. 13)

Ein wichtiger Bestandteil von Hive ist der Metastore. Im Metastore werden verschiedene Metadaten zu Tabellen gespeichert, die es unter anderem ermöglichen, dass die mit Hive abgespeicherten Daten durch ein relationales Datenbankmanagementsystem verwaltet werden können. (vgl. [Luber et al. (2017)])

Abbildung 10 zeigt vereinfacht, wie ein Prozess unter der Verwendung von Hive aussieht:

- Hive kompiliert die Abfrage
- Tez führt die Abfrage aus
- YARN weist die für die Applikation benötigten Ressourcen im gesamten Cluster zu und schaltet Hive Jobs für YARN-Abfragen frei
- Je nach Tabellentyp, passt Hive die Daten entweder in HDFS oder im Hive Warehouse an bzw. fragt jene ab
- Hive gibt die Abfrageergebnisse bspw. durch eine JDBC (Java Database Connectivity) Verbindung aus

(vgl. [Cloudera, Inc. (o. J.)])

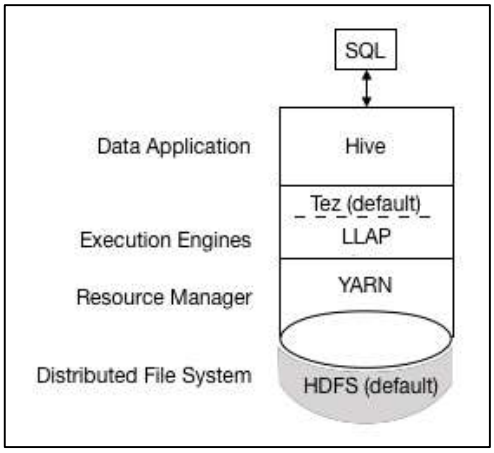


Abbildung 10 Vereinfachte Darstellung eines Prozesses unter Verwendung von Hive (vgl. [Cloudera, Inc. (o. J.)])

Apache HBase

HBase ist eine quelloffene, nicht relationale, verteilte Datenbank und baut auf dem HDFS auf. Dabei wurde HBase nach dem Vorbild von Google's Bigtable modelliert. (vgl. [Chang et al. (2006)]). Tabellen werden in HBase als spaltenorientierte Schlüssel-Werte Datenspeicher realisiert, wobei immer nur eine Spalte die Primärschlüssel enthalten kann. Alle Zugriffe auf die Zeile einer HBase-Tabelle müssen somit über den Primärschlüssel erfolgen. Daten werden in HBase immer vom Typ Byte gespeichert. Die Versionierung von Daten erfolgt über Timestamps von Typ Long Integer. HBase sortiert dabei in absteigender Reihenfolge, damit beim Lesen eines Datensatzes der aktuellste Eintrag zuerst gefunden wird. (vgl. [Apache HBase Team (2020)]).

Das Datenmodell von HBase lässt sich in sechs Konzepten beschreiben:

- Table: Die Daten werden in einer Tabelle zusammengefasst.

- Row: Unter Row versteht man eine Zeile in einer Tabelle. Jede Zeile kann durch einen eindeutigen Schlüssel identifiziert werde.
- Column family: Eine column family beschreibt die Gruppierung von Daten innerhalb einer Zeile.
- Column qualifier: Der column qualifier identifiziert die einzelnen Spalten aus einer column family.
- Cell: Beschreibt eine Zelle, die Daten speichern kann.
- Version oder Timestamp: Die Werte innerhalb einer Zelle können verschiedene Versionen haben. Der Timestamp dient dazu diese zu identifizieren.

(vgl. [Rossy (2015)])

RowKey (Primärschlüssel)	Timestamp	Column Family (Werte 1) Bsp. Namen		Column Family (Werte 2) Bsp. Alter
100	T1	Vorname		38
100	T2	Vorname	Nachname	
101	T3	Vorname		22

Abbildung 11 HBase Tabellenaufbau
(Eigene Darstellung in Anlehnung an [Rossy (2015)])

Durch die spaltenorientierte Darstellung und das lose Schema hat HBase beispielsweise den Vorteil der einfachen Skalierbarkeit und die Möglichkeit, große Datenmengen zu speichern. Auswertungen oder Abfragen mittels SQL sind dadurch jedoch nicht möglich.

4.2 Technologische Basis

Einige Elemente der möglichen Konzepte und Programme zur Datenablage finden sich bei allen drei Arten der Datenspeicherung im HDP-Cluster wieder, z. B. die Kommunikation mit der MES Hydra API. Auf diese soll hier eingegangen werden.

Apache NiFi

Apache NiFi bietet ein effizientes und zuverlässiges System zur Verarbeitung und Verteilung von Daten an (vgl. [Apache NiFi (o. J.)]). Um die Funktionalitäten von NiFi nutzen zu können, wurde der Service auf dem HDP-Cluster der DHBW Mosbach installiert. Somit kann mittels Web-GUI (Graphical User Interface) auf NiFi zugegriffen und Prozesse definiert werden.

NiFi bietet eine grafische Oberfläche (Table), auf der z. B. verschiedene Prozessoren platziert werden können. Diese bieten unterschiedlichste Funktionen zur Datenverarbeitung an. Für diese können außerdem In- und Outputs definiert und so der Datenfluss exakt bestimmt werden.

Die Anbindung der API über Apache NiFi wird mittels des "InvokeHTTP"-Prozessors ermöglicht. Um eine einfachere Wiederverwendbarkeit zu gewährleisten wird eine Variable für die API-URL festgelegt,

welche für Testzwecke auf die Test-API-URL und für den Produktivbetrieb auf die Produktions-API-URL geswitcht werden kann. Ein repräsentativer API-Call wird in Abbildung 12 dargestellt.

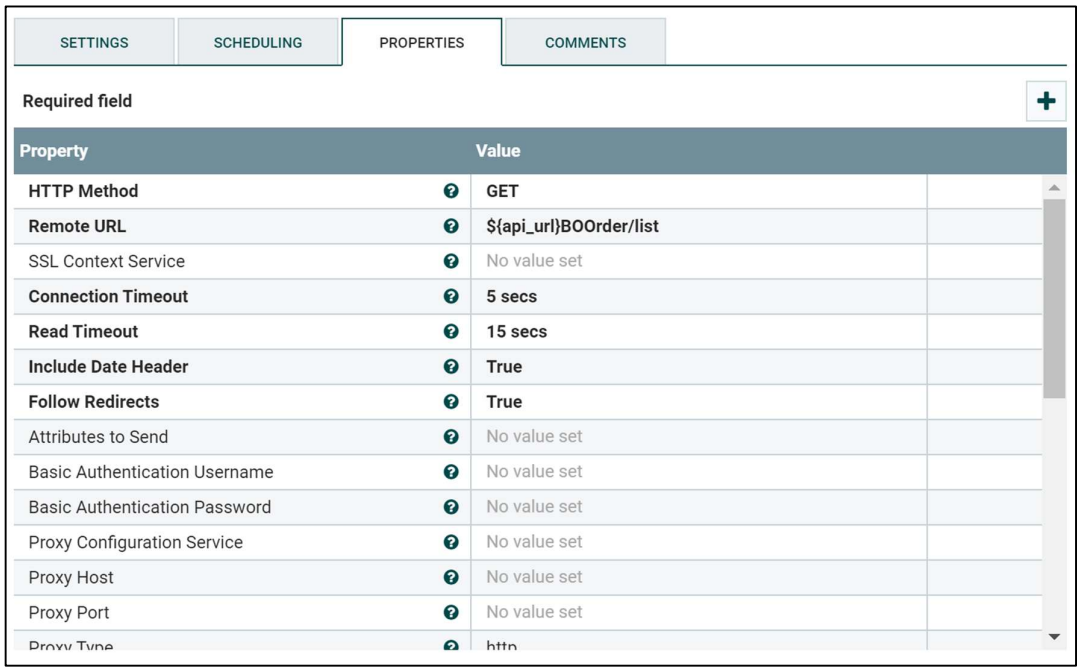


Abbildung 12 BOOrders API-Call (NiFi)

Eigenentwicklung in Python

Die Eigenentwicklung der Datenablage in HDFS, Hive und HBase soll in der Programmiersprache Python umgesetzt werden. Verwendet wird Version 3.8.2.

Für Python existiert mit dem Python Package Index (PyPI) eine umfangreiche Bibliothek von Modulen, die nach erfolgreicher Installation in Python-Skripten genutzt werden können. Unter anderem sind hier Module für die Anbindung einer API und die Kommunikation mit HDFS, Hive und HBase vorhanden (vgl. PyPI-Suche: [Python Software Foundation (2020)]). Zur Installation der Module bietet PyPI einen Paketmanager (pip), der den Download und die Installation der für das Modul notwendigen Dateien übernimmt (vgl. [PyPA (2020)]). Um ein Python Skript auf einem Rechner ausführen zu können, müssen zuvor alle im Skript verwendeten Module (erkennbar am *import*-Befehl in einem Python-Skript) installiert werden.

Die Anbindung der MES Hydra REST API erfolgt in den entwickelten Prototypen über das Python-Modul *requests* in der Version 2.23.0. Mithilfe der Antwort des API-Endpunktes *<baseURL>/meta* wird eine Liste mit allen abzurufenden Domains und Services erstellt. Diese Liste wird in einer *for*-Schleife iteriert. Pro Iteration wird die URL zum API-Abruf dynamisch aus den Angaben zu Domain und Service in der Liste erstellt. Die URL orientiert sich am Schema *<baseURL>/meta/<Domain>/<Service>*. Abbildung 13 zeigt den Programmcode, der die API-Anfrage durchführt. Die "headers" enthalten Autorisierungsinformationen für die API. Der Request-

Timeout wird auf 60 Sekunden gesetzt, um eine zu lange Laufzeit des Skriptes bei einer nicht antwortenden API zu vermeiden.

Somit werden die Daten nacheinander für alle Endpunkte mit "list…"-Services abgerufen und können auf dem HDP-Cluster gespeichert werden.

```
for entry in allDomainsAndServices():
    # send request, get response
    url_data = api_url + "/data/" + str(entry[0]) + "/" + str(entry[1])
    response_data = requests.request("GET", url_data, headers=headers, timeout=60)
```

Abbildung 13 Python-Skript Ausschnitt zur API-Anfrage

Da die API JSON-Antworten liefert, wird die Weiterverarbeitung in den Python-Skripten mittels des Moduls *json5* (Version 0.9.4) durchgeführt. Die Daten werden im Folgenden aus den JSON-Gerüsten extrahiert und mithilfe von zweidimensionalen Python-Listen in eine Tabellenform gebracht. Zur Umwandlung in CSV-Dateien wird auf das Modul *numpy* (Version 1.18.4) zurückgegriffen. Dieses bietet mit *savetxt* eine Funktion zur Generierung einer CSV-Datei aus einer Python-Liste.

Nach jedem Aufruf eines API-Endpunktes liefern die Skripte eine Meldung zum Status der abgespeicherten Domain zurück. "Domain gespeichert" wird bei der erfolgreichen Ablage auf dem HDP-Cluster zusammen mit der aktuellen Anzahl der im Skriptdurchlauf abgelegten Domains mitgeteilt.

Wird bei einer Domain von der API eine Antwort vom Datentyp "ERROR" zurückgegeben, wird das im Log angegeben und die Domain übersprungen. Dies ist beispielsweise bei "list…"-Services der Fall, die eine Parameterübergabe erwarten. Um von diesen ebenfalls Daten abzurufen, müssten betreffende Services manuell inspiziert, die Parameter festgestellt und deren Werte festgelegt werden. Daher wird die Abfrage der API in dieser Seminararbeit auf die Domains und Services beschränkt, die ohne Parameterangabe Daten zurückliefern.

Einige API-Endpunkte senden eine leere Antwort (*[ ]*). Dies wird im Log mit "leer" gekennzeichnet. Zusätzlich wird jeweils der Name der Domain, der Name des Services und ein Zeitstempel ausgegeben.

Weitere in den Programmen genutzte Module sind *datetime* (zur Ermittlung und Verarbeitung von Speicherdatum und --zeit) und *sys* (verwendet wird hier eine Funktion zum vorzeitigen Beenden eines Skriptes). Für die einzelnen Datenablagearten sind weitere Module notwendig, auf die in den folgenden Kapiteln je im Bereich Eigenentwicklung in Python eingegangen wird.

4.3 Konzeption der Datenablage im HDFS

Ein essenzieller Baustein der Ablage der Daten im HDFS ist das zu verwendende Dateiformat. Eine Ablage der unveränderten API-Antworten im JSON-Format ist möglich. Wie bereits bei der Sichtung der MES Hydra API festgestellt, erinnert der Aufbau der API-Antwort (Standardkonfiguration) an eine Tabelle.

Zur Ablage in einem Dateisystem bieten sich daher CSV-Dateien mit folgendem Aufbau an: Die erste Zeile enthält die Spaltenbeschriftungen; alle weiteren Zeilen bilden die einzelnen Datensätze ab. Um alle Informationen aus den API-Daten zu übernehmen, wird der Datentyp im Spaltennamen ergänzt. Als Trennzeichen wird in der CSV-Datei das in Deutschland übliche Semikolon verwendet. Strukturierte Daten im CSV-Format eignen sich gut für die Weiterverarbeitung, nachdem die Daten auf dem HDP-Cluster gespeichert wurden. Daher wird diese Option gegenüber der Ablage als JSON-Datei präferiert.

Der Dateiname der CSV-Datei sollte sich aus dem Domainnamen (zur Identifikation der einzelnen Domains) und Ablagedatum und -zeit zusammensetzen. Zusätzlich ist der abgerufene Service hilfreich, da manche Domains mehrere 'list…'-Services haben. Durch das Datum können die Datenbestände der API zu verschiedenen Zeitpunkten rekonstruiert und verglichen werden. Hierbei kann je nach Präferenz und späterer Nutzung der abgelegten Daten an erster Stelle der Name und an zweiter Stelle das Datum oder an erster Stelle das Datum und an zweiter der Name gesetzt werden. Je nach Reihenfolge können die Dateien nach Domain oder nach Datum sortiert werden. Im Folgenden wurde Variante eins gewählt, kann jedoch angepasst werden. Außerdem wäre es möglich, die Dateien nach Namen oder Datum in einzelnen Ordner zu gruppieren.

Apache NiFi

Theoretisch ist es in NiFi möglich, die JSON-Datei, die die MES Hydra API bereitstellt, direkt in HDFS zu speichern. Dazu werden lediglich zwei Prozessoren benötigt. Der erste Prozessor "InvokeHTTP" macht den API-Aufruf und leitet das Ergebnis weiter an den zweiten Prozessor "PutHDFS", welcher die Datei im HDFS Cluster abspeichert, siehe Abbildung 14.

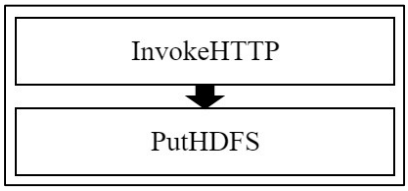


Abbildung 14 Abspeicherung der JSON-Datei im HDFS

Sollen die Daten allerdings als CSV-Dateien im HDFS gespeichert werden, sind einige zusätzliche Prozessoren nötig.

Zunächst wird die API durch einen "InvokeHTTP" Prozessor, wie im oben genannten Fall abgerufen. Das Ergebnis wird anschließend an einen "SplitJSON" Prozessor, welcher die JSON-Datei in Zeilen aufteilt, übergeben. Daraufhin folgt ein "EvaluateJSONPath" Prozessor, welcher die JSON-Datei evaluiert, damit die Datei nach dem vorangegangenen Prozess weiterverarbeitet werden kann. Es folgt ein weiterer "SplitJSON" Prozessor, welcher die Data-Attribute (also die eigentlichen Daten) von der restlichen JSON-Datei abschneidet. Im nächsten Schritt trennt ein "RouteOnAttribute" Prozessor die Daten von der ersten Zeile, welche nur die Benennungen und Typen der einzelnen Werte beinhaltet.

An dieser Stelle werden die gleichen Daten an verschiedene Prozessoren weitergegeben. Einmal zum Abspeichern der Daten als CSV-Datei im HDFS und zum anderen zum Abspeichern in Hive. In diesem

Abschnitt wird das Abspeichern der Daten in Form einer CSV-Datei im HDFS behandelt. In Abschnitt "4.4 Konzeption der Datenablage in Hive" folgt anschließend das Konzept zum Abspeichern der Daten in Hive.

Im Falle von HDFS werden die Daten im folgenden Schritt von einem "ReplaceText" Prozessor aufbereitet, indem überflüssige Klammern entfernt werden. Anschließend dazu werden die zuvor in Zeilen aufgeteilten Daten in einem "MergeContent" Prozessor wieder zu einer Datei zusammengefügt und den im Schema festgelegten Spalten – den Überschriften der späteren CSV-Datei (entsprechen theoretisch der ersten Zeile der ehemaligen JSON-Datei) – zugeordnet. Im vorletzten Schritt werden die nun zusammengefügten Daten über den Prozessor "UpdateAttribute" in eine CSV-Datei konvertiert und nach der derzeitigen Serverzeit benannt. Als Letztes folgt das Abspeichern der Datei im HDFS Cluster mit dem Prozessor "PutHDFS". Abbildung 15 veranschaulicht den beschriebenen Prozess.

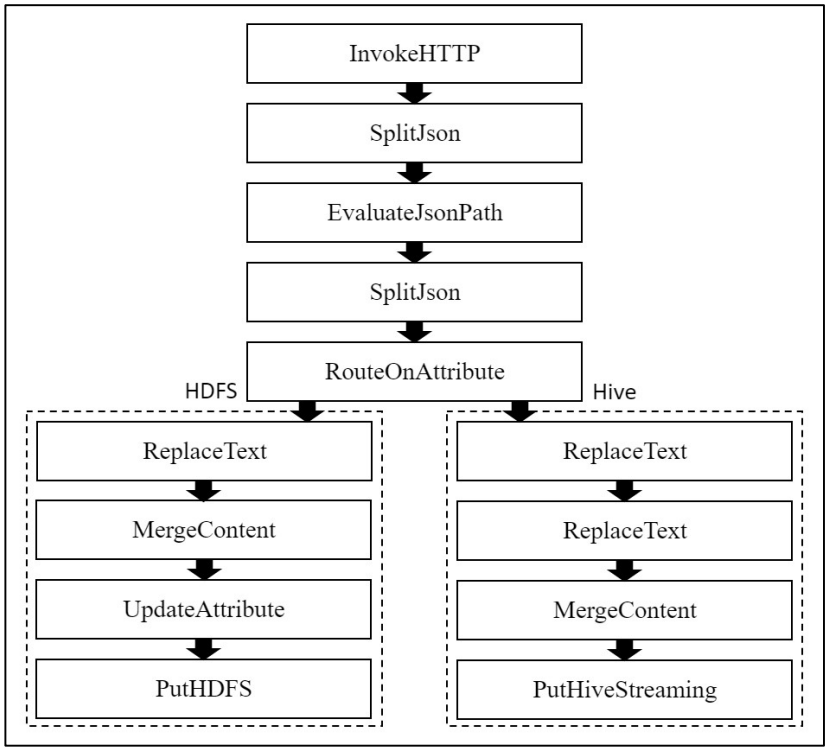


Abbildung 15 Abspeichern der Daten in Tabellenform in HDFS und Hive

Theoretisch ist es möglich, die erste Zeile der JSON-Datei so zu verarbeiten, dass diese für die CSV-Datei als Überschrift verwendet werden könnte. In diesem Fall wären allerdings viele zusätzliche, zudem sehr fehleranfällige, Schritte nötig. Hinzu kommt, dass die Tabelle für Hive vor dem Abfragen und Verarbeiten in NiFi zunächst erst erstellt werden muss (siehe Abschnitt 4.4) und das Schema deshalb bereits vorliegt.

Eigenentwicklung in Python

Für den Zugriff auf das HDFS wird das Python-Modul *hdfs* (Version 2.5.8) verwendet. Mithilfe des *InsecureClients* wird eine Verbindung zum HDFS hergestellt. Das Python-Modul nutzt die WebHDFS

REST API. Diese ist unter Port 50070 erreichbar und stellt HTTP-Methoden zum Zugriff auf das HDFS bereit. Das WebHDFS ist somit mit allen Programmiersprachen, die REST API-Abrufe ermöglichen, aufrufbar.

Das Python-Modul *hdfs* stellt wiederum Funktionen bereit, die die Kommunikation mit der API abwickeln. Die Funktion *upload* beispielsweise ermöglicht das Hochladen einer Datei vom lokalen Dateisystem auf einen bestimmten Pfad des HDFS. Die Funktion *write* kann genutzt werden, wenn beispielsweise eine im Python-Skript erstellte und nicht auf dem lokalen Dateisystem gespeicherte Datei auf dem HDFS abgespeichert werden soll. So auch im Skript zur Speicherung der MES Hydra REST API-Daten (im Ordner HDFS des GitHub-Repositories). Die Daten je Domain der REST API werden im Programmablauf in einer Python-Liste zusammengefügt. Abbildung 16 zeigt den Teil des Skriptes, der für die anschließende Speicherung zuständig ist. Der Speicherpfad im HDFS ist auf '/tmp/data/WI17A_MES/python' gesetzt.

```
with client_hdfs.write(hdfs_path, encoding = 'utf-8', overwrite=True) as writer:
    np.savetxt(writer, data_toSave, delimiter=";", fmt='%s')
```

Abbildung 16 HDFS-Skript Ausschnitt zur Speicherung der Dateien auf dem HDFS

4.4 Konzeption der Datenablage in Hive

Zur Ablage der Daten in Hive werden interne Tabellen genutzt. Pro Domain erfolgt die Anlage einer Tabelle, die Spaltennamen werden mithilfe der Metadaten der API-Antwort erstellt. Der in den Metadaten angegebene Datentyp kann in fast allen Fällen für die Hive-Tabellenanlage übernommen werden. Lediglich der Datentyp *DATETIME* muss durch den in Hive verwendeten Datentyp *TIMESTAMP* ersetzt werden. Die Daten bedürfen jedoch keiner Anpassung.

Bei jeder Ausführung der NiFi-Routine oder des Python-Skripts sollen die aktuell von der API bereitgestellten Daten in die Tabelle der entsprechenden Domain hinzugefügt werden. Um die Einträge pro Tabelle differenzieren zu können, wird bei jedem Datensatz eine Spalte *entrytimestamp* ergänzt, die das Datum der Ablage der Datensätze einer Domain in die Hive-Tabelle enthält. So gibt es auch hier die Möglichkeit, die einzelnen Stände der API zu verschiedenen Zeitpunkten zu rekonstruieren und zu vergleichen.

Apache NiFi

Um die Daten der MES Hydra API über NiFi in Hive abzulegen, müssen diese über einen Prozessor abgefragt werden. In Abschnitt 4.3 "Apache NiFi" wird der Ablauf der NiFi-Routine bereits beschrieben, der bis Prozessor "RouteOnAttribute" gleich ist. Dort verzweigt sich der Ablauf. (siehe Abbildung 15). In den zwei aufeinander folgenden Prozessoren "ReplaceText" wird die schließende eckige Klammer ([ ) durch ein Leerzeichen ersetzt und die öffnende eckige Klammer ([ ) durch den einen Zeitstempel "entrytimestamp" der anzeigt wann der Datensatz in die Hive Datenbank gespeichert

wurde. Darauf folgt ein MergeContent Prozessor der diese beiden Ergebnisse wieder zusammenfasst. Im Prozessor "PutHive3Streaming" werden die durch Kommata getrennten Werte in Hive geschrieben. In diesem Prozessor muss dafür ein sog. Record Reader angelegt werden, der den Aufbau der JSON beschreibt. Zur Interpretation dieser, wird ein Avro-Schema benötigt, welches festlegt, welche Daten in welche Spalte der Datenbanktabelle gespeichert werden. Das Avro Projekt ist ebenfalls von der Apache Software Foundation und ist ein Remote-Procedure-Call- und Serialisierungs-Framework. Es wird zur Definition von Datentypen und Protokolle mithilfe von JSON-Dateien verwendet. Innerhalb dieses Schemas wird die oben angesprochene Änderung des Datentypen DATETIME (API-Antwort) zu TIMESTAMP (Hive) festgelegt.

Somit ist schnell ersichtlich, wenn einer der Prozessoren einen Output liefert, welcher nicht den gewünschten Spezifikationen des nächsten Prozessors entspricht. Die Datenbanken und Tabellen die über NiFi zur Speicherung verwendet werden, müssen vor der Ausführung der NiFi Routine auf dem Datenbank Server angelegt werden. Dazu verbindet man sich per SSH auf das Cluster der DHBW und spricht die Datenbankkonsole über einen beeline-Befehl an (siehe Abbildung 17):

```
beeline - u 'jdbc:hive2://localhost:10000' -n hive
```

Abbildung 17 beeline-Befehl

Nun kann man mit SQL-Befehlen die Datenbanken und gewünschten Tabellen anlegen. Für die im Test verwendete Tabelle BOOrders_data wurde der folgende CREATE-Befehl verwendet (siehe Abbildung 18):

```
CREATE TABLE BOOrders_data (
    entrytimestamp STRING,
    order_cycle_target_format STRING ,
    order_pulse_factor_target DECIMAL ,
    [...]
    order_external_processing_indicator STRING ,
    order_is_archived STRING
);
```

Abbildung 18 Ausschnitt aus dem CREATE-Befehl

Eigenentwicklung in Python

Zur Kommunikation mit Hive wird das von Mitarbeitern der Dropbox Inc. entwickelte Python-Modul *pyhive* (Version 0.6.2) genutzt. Dieses Modul wiederum verwendet die Module *sasl* (Version 0.2.1), *thrift* (Version 0.13.0) und *thrift-sasl* (Version 0.4.2). Thrift (Apache Thrift) umfasst und beschreibt unter anderem das Protokoll und die Schnittstelle, auf deren Basis die Kommunikation mit Hive stattfindet. SASL (Simple Authentication and Security Layer) ist ein Standard für die Authentifizierung auf Protokollbasis, der bei Thrift verwendet wird. Gegebenenfalls kann eine Installation von Softwarepaketen für das Betriebssystem zur Unterstützung von SASL notwendig sein (bei Ubuntu beispielsweise *libsasl2-dev*).

Bei der Entwicklung des Skriptes zur Abspeicherung in Hive wurden mehrere mögliche Wege getestet. In allen Varianten werden die Metainformationen pro Domain beim erstmaligen Ausführen des Skriptes zur Anlage der Tabellen genutzt. Bei allen weiteren Ausführungen des Skriptes werden die neuen Daten in die bestehenden Tabellen hinzugefügt. Hierbei wird die Funktion *execute* des Moduls *pyhive* zur Ausführung von an die SQL-Syntax angelehnte HQL-Statements genutzt. Diese Statements werden im Skript dynamisch erstellt.

Die Varianten unterscheiden sich in der Art des Einfügens der Daten in die Hive-Tabellen. Variante eins fügt jeden Datensatz mit einem eigenen *Insert*-Statement ein. Bei einigen Tests während der Entwicklung fiel hierbei die Dauer von ca. 3-5 Sekunden pro Datensatz auf.

Variante zwei nutzt pro Domain ein *Insert*-Statement. Hierdurch konnte die Dauer in den Tests auf ca. 0,1-0,3 Sekunden pro Datensatz verkürzt werden.

In der dritten Variante werden die Daten vor dem Einfügen in die Hive-Tabelle auf dem HDFS zwischengespeichert. Hierzu werden die Daten, wie bereits im Skript zur Speicherung auf dem HDFS, in einer zweidimensionalen Python-Liste zusammengeführt und als CSV ohne Spaltenbezeichnungen auf dem HDFS gespeichert. Darauffolgend werden die Daten mittels eines HQL-Statements in eine zuvor angelegte, im Textfile-Format gespeicherte Hive-Tabelle mit dem Namen *hive_tt* geladen. Dieses Format ist notwendig, um das Einfügen von Daten aus einer CSV-Datei zu ermöglichen. Im letzten Schritt werden alle in der Tabelle *hive_tt* vorhandenen Daten mittels *Insert*-Befehl in die reguläre Hive-Tabelle der Domain geladen und die zur Zwischenspeicherung notwendige Tabelle und Datei wieder entfernt. Die Dauer pro Datensatz befand sich in den Tests bei 0,1 Sekunden. Aus diesem Grund wird im finalen Python-Skript zur Ablage in Hive diese Vorgehensweise verwendet. Es sind jedoch die Skripte für alle drei Varianten im GitHub-Repository im Ordner Hive zu finden, wobei die Entwicklung von Variante eins nach der Entdeckung von Variante zwei und drei eingestellt wurde.

4.5 Konzeption der Datenablage in Apache HBase

Für die Datenablage in HBase wurde ebenfalls ein Konzept ausgearbeitet. Hierbei wird der gesamte Datenbestand in einer einzigen Tabelle gehalten. Diese Tabelle enthält eine Spalte für den Primärschlüssel sowie eine automatisch angelegte Spalte für den Zeitpunkt der Speicherung.

Des Weiteren beinhaltet die Tabelle eine column family für jede abgefragte Domain, in der sich je eine Spalte pro Klasse der Domain befindet. Eine schematische Darstellung der Tabelle liefert Abbildung 19.

Primär-schlüssel	BOOrder				BOResource				MDWorkplanOrder				…
	Attribut 1	Attribut 2	…	Attribut n	Attribut 1	Attribut 2	…	Attribut n	Attribut 1	Attribut 2	…	Attribut n	…
…	…	…	…	…	…	…	…	…	…	…	…	…	…

Abbildung 19 Beispiel einer HBase Tabelle

HBase NiFi

Für die Ablage der Daten in HBase über NiFi muss die HBase Tabelle zuvor mit vordefinierter Struktur mittels CREATE-Befehl angelegt werden. Analog zu der Ablage in HDFS über Nifi wird zunächst die API von MPDV-Hydra über einen Prozessor angesprochen und die Antwort an einen weiteren Prozessor weitergereicht um die JSON-Datei in die entsprechende HBase Tabelle zu schreiben (Abbildung 20).

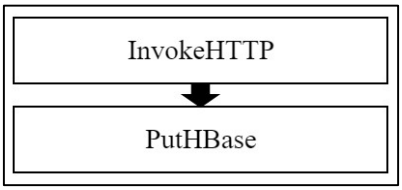


Abbildung 20 Abspeicherung der Datei in HBase

Um die gewünschte Einteilung der column familys und column qualifiers zu erreichen, wird die JSON wie beim Ablegen in HIVE oder als CSV-Datei in HDFS zunächst in die in Kapitel 4.3 erläuterte Form gebracht. Danach wird der Prozessor PutHBaseCell (Abbildung 21) genutzt, um die Daten an die richtige Stelle in der HBase Tabelle zu schreiben.

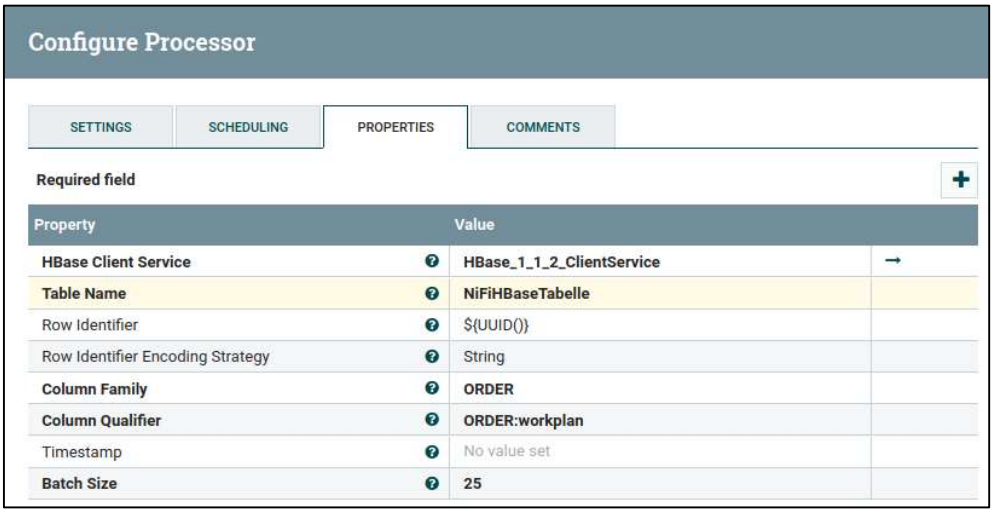


Abbildung 21 HBasePutCell für BOOrders:workplan (NiFi)

Eigenentwicklung in Python

Für die Anbindung von Python-Programmen an HBase ist das Paket *happybase 1.2.0* nötig, sowie das Starten eines Thrift-Servers über SSH. Die Herstellung einer SSH-Verbindung mit Python wird durch das Paket *paramiko 2.7.1* ermöglicht.

Die relevanten Daten werden aus der API-Antwort extrahiert und in der HBase Tabelle in die korrespondierenden Spalten geschrieben. Hierbei geht der Datentyp verloren, da HBase Daten grundsätzlich als Byte-Array abspeichert. Daher ist auch keine Konvertierung erforderlich.

4.6 Evaluation der Datenablagearten

Nach der Konzeption und Ausarbeitung verschiedener Ablagearten lassen sich bereits einige vorläufige Ergebnisse feststellen.

Zum einem wird die Ablage in CSV-Dateien für die abzuspeichernden Daten als geeignet erachtet, da die von der API zurückgelieferten Daten (Standardkonfiguration) eine tabellenähnliche Struktur aufweisen (siehe auch Kapitel zur API-Sichtung und Kapitel 4.3).

Zum anderen zeigt sich die Ablage der Daten in HBase aus verschiedenen Gründen als ungeeignet. Beispielsweise ist HBase für unstrukturierte Daten konzipiert. Es wird dementsprechend keine Typbindung unterstützt. Zudem eignet sich HBase eher für sehr große Datenpakete (im Bereich von mehreren Petabytes). Die größten bei Tests entstandenen Dateien umfassten dahingegen nur einige Megabytes. Hinzu kommt außerdem, dass Daten, die in HBase gespeichert werden, lediglich nach dem Primärschlüssel durchsucht werden können, welcher bei der Ablage gewählt wird. Dies schränkt die Möglichkeiten der späteren Auswertung der Daten ein.

Bei den Performancetests in Kapitel 5 werden deshalb nur noch die Lösungen zur Ablage in Hive und HDFS berücksichtigt. Zudem hebt sich die Eigenentwicklung in Python gegenüber der Standardlösung in NiFi ab, da sie höhere Flexibilität für spätere Anpassungen aufweist. Somit lässt sich das Programm besser auf die MES Hydra API anpassen. Bei der Python Entwicklung kann beispielsweise dynamisch jeder API-Endpunkt abgefragt werden, während im Falle der NiFi Lösung erhebliche Anpassungen für jede der API-Adressen vorgenommen werden müssten. (Unter anderem das manuelle Erstellen der Tabelle für Apache Hive, Anpassung der API-Adresse beim Prozessor "InvokeHTTP" und das manuelle Erstellen eines Avro-Schemas.)

Ein Nachteil der Eigenentwicklung ist jedoch, dass vor Inbetriebnahme erst die Einrichtung erfolgen muss (z. B. Installation der benötigten Module). Sobald dies erledigt ist, überwiegen allerdings die Vorteile der Eigenentwicklung in Python gegenüber der Standardlösung in NiFi.

Unabhängig davon lässt sich eine Tendenz in Richtung der Ablage der MES Hydra API-Daten in Hive-Tabellen anstelle der Ablage im HDFS erkennen. Dies hat primär zwei Gründe. So hat HDFS den Nachteil des "Small Files Problem", aufgrund dessen HDFS nicht für viele kleinere Dateien - wie sie auch bei der API vorliegen - geeignet ist (vgl. [Balint (2009)]). Im Gegenzug dazu, bietet Hive zusätzlich den Vorteil gegenüber HDFS, dass SQL-ähnliche Abfragen (HQL), und damit die direkte Weiterverarbeitung und Auswertung der Daten, möglich sind.

Auch wenn sich durch die beiden genannten Vorteilen von Hive gegenüber HDFS bereits eine präferierte Lösung herauskristallisiert, sollen dennoch beide Möglichkeiten im folgenden Performancetest je in der Umsetzung mit Apache NiFi und der Eigenentwicklung in Python betrachtet werden.

5 Performancetests

Für den Performancetest wird die Domain BOOrders der MES Hydra API herangezogen. Hierbei handelt es sich um eine repräsentative Domain mit 2071 Datensätzen (Produktions-API, Port 8080); jeder Datensatz umfasst 548 Spalten (Stand 2020-05-27).

Durchgeführt wird der Performancetest für die erstellte NiFi-Routine und die entwickelten Python-Skripte zur Ablage der Daten je im HDFS und Hive. Betrachtet werden Stabilität und Zuverlässigkeit, Geschwindigkeit und Skalierbarkeit.

Stabilität und Zuverlässigkeit

NiFi: Bei den Performancetests zeigte sich, dass die Stabilität bei aktueller Konfiguration des Services nicht zufriedenstellend ist. Der zugeordnete maximal verfügbare RAM (2048 MB) war beim Testlauf mit der Produktions-API voll ausgelastet, was zum Absturz von NiFi führte. Auch ein zugewiesener Arbeitsspeicher von 4096 MB brachte keine erkennbare Verbesserung. Die Installation von NiFi auf einem separaten Cluster, mit entsprechender Ausstattung und direktem Zugriff auf das HDP-Cluster, könnte dieses Problem beheben, müsste jedoch getestet werden.

Python: Die Python-Skripte werden direkt auf dem HDP-Cluster-Server ausgeführt. Daher steht die freie Serverkapazität zur Verfügung und die Skripte laufen stabil. Bei den Tests konnten keine Abstürze verzeichnet werden.

Geschwindigkeit

Im Bereich Geschwindigkeit wird die zeitliche Dauer der Ausführung der NiFi-Routine und der Python-Skripte gemessen. Aufgrund des unter Stabilität beschriebenen Problems wird bei NiFi der Test hier nur mit einigen Daten der Produktions-API durchgeführt, Python greift auf alle Daten der Produktions-API zurück (jeweils Domain BOOrders). Die Zeit wird wie folgt gemessen:

NiFi: Pro Prozessor wird die benötigte Zeit nach Abschluss der Ausführung angegeben. Diese Zeiten werden von allen Prozessoren addiert und so die benötigte Dauer ermittelt (siehe Abbildung 22 links).

Python: Zum Start und Ende der Skriptausführung auf dem HDP-Cluster-Server wird jeweils die aktuelle Zeit auf dem Terminal ausgegeben und im Anschluss die Differenz zwischen beiden Zeiten ermittelt (siehe Abbildung 22 rechts).

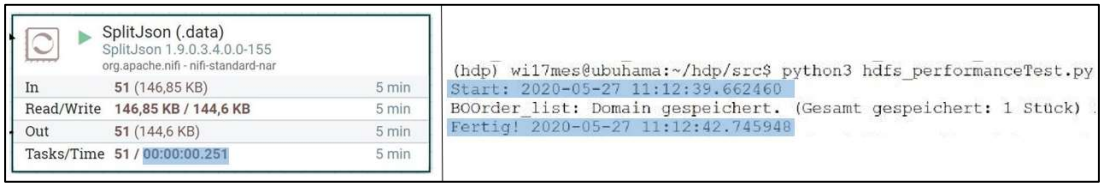


Abbildung 22 Zeitmessung in NiFi und Python

Aufgrund der anfänglichen Probleme mit der Stabilität von NiFi wurde die Ablage zuerst mit wenigen Datensätzen getestet und später auf alle Datensätze ausgeweitet und jeweils die Dauer erfasst. Eine Übersicht hierzu ist in Anhang A1 bis A4 zu finden.

Die Erfassung der Ablagedauer wurde zur Dokumentation in der Seminararbeit einmalig vorgenommen. Einige weitere Tests zeigten, dass eine erneute Durchführung davon geringfügig abweichende Werte ergeben könnte, da die Dauer auch durch Faktoren wie z. B. die aktuelle Serverauslastung oder das Datenverkehrsaufkommen im Netzwerk beeinflusst wird.

Abbildung 23 zeigt die erfassten Dauern für die Ablage aller Datensätze der BOOrders (Produktions-API). Hierbei ist zu erkennen, dass bei beiden Ablagearten Python schneller ist. Bei der Ablage im HDFS beträgt die Abweichung 21,408 Sekunden, bei der Ablage in Hive 13,643 Sekunden.

Allerdings sind die Ablagearten HDFS und Hive nicht direkt vergleichbar, da die Daten in anderen Formen abgelegt werden und beispielsweise die Umformung der Daten auf das Hive-Schema länger dauert als die einfache Ablage im HDFS. Jedoch können die Daten in Hive-Tabellen im Anschluss besser verarbeitet werden z. B. aufgrund der Möglichkeit von SQL-ähnlichen Abfragen.

Python kann allerdings gegenüber NiFi im Bereich zeitliche Dauer der Ausführung verglichen werden. Hier zeigt sich, dass die Verarbeitung mit den Python-Skripten deutlich schneller ist.

	Apache NiFi *Produktiv-API 2071 Datensätze*	**Python** *Produktiv-API 2071 Datensätze*
HDFS	24,491	3,083
Apache Hive	44,879	31,236

Abbildung 23 Tabelle Übersicht Dauern (Angabe in Sekunden)

Testweise wurden die Python-Skripte zur Ablage der Daten im HDFS und Hive auf allen Endpunkten der Produktions-API ausgeführt. Erreicht und abgelegt wurden hierbei die Daten von 241 API-Endpunkten. Die Dauer des HDFS-Skriptes lag bei ca. sieben Minuten. Die Ausführung des Hive-Skriptes dauerte ca. 18 Minuten.

Skalierbarkeit

NiFi: Um die NiFi-Routine auf alle API-Endpunkte auszuweiten, muss für jede Domain manuell die Header-Zeile für die Ablage in HDFS angepasst, ein eigenes Avro-Schema erstellt und die Hive-Tabelle angelegt werden. Die Header-Zeile für HDFS kann auch dynamisch erstellt werden, dies benötigt jedoch entsprechende Ressourcen. Daher ist der Punkt Skalierbarkeit bei NiFi mit einem hohen manuellen, mit Anzahl der Domains linear steigendem Aufwand verbunden.

Python: Hier können die Meta-Informationen zu Domains und Services automatisch ausgelesen und auf dieser Basis die Informationen zur Gestaltung der CSV-Header zur Ablage im HDFS und Hive-Tabellen angelegt werden. Die einmalige Definition der Logik ist auf die weiteren Domains ausweitbar (siehe Skripte zur Ablage aller Domains).

Aktuell werden nur "list…"-Services berücksichtigt, die keine Parameterübergabe erwarten. Sowohl bei den Hive-Routinen als auch in den Python-Skripten müssen bei der Erweiterung um "list…"-Services, die obligatorische Parameter umfassen, diese manuell ermittelt, mit Werten versehen und implementiert werden.

6 Schlussbemerkungen

Im Rahmen dieser Seminararbeit wurden mögliche Datenablagearten im HDP-Cluster eruiert und mehrere für die bereitgestellten Daten mögliche Lösungen entwickelt.

Hierzu wurde die MES Hydra REST API gesichtet und festgestellt, dass sich für die Daten eine tabellenartige Struktur eignet. Als mögliche Ablagearten im HDP-Cluster wurden HDFS, Apache Hive und Apache HBase eruiert. Es erfolgte die Konzeption der Speicherung der Daten in den jeweiligen Ablagearten. Hierfür wurde ein Konzept mit der Nutzung von Apache NiFi entwickelt und eine Eigenentwicklung in der Programmiersprache Python angegangen, um die Möglichkeiten von Marktlösung und Eigenentwicklung gegenüberzustellen.

Es zeigte sich, dass die Ablagearten HDFS und Apache Hive gegenüber Apache HBase präferiert wurden, da sich diese besser für die abzulegenden strukturierten Daten eigneten. Anschließend erfolgten Performancetests für diese beiden Ablagearten, jeweils für die Lösungen in NiFi und Python.

Auf Grundlage der in der Seminararbeit gewonnenen Erkenntnisse, ist NiFi für die abzulegenden Daten nicht zu empfehlen. Vor allem für die Ausweitung auf die Ablage aller in der API vorhandenen Domains sind viele manuelle Tätigkeiten notwendig z. B. Anlage eines Avro-Schemas und das Anlegen der Datenbank-Tabellen über SSH.

Stattdessen eignen sich die im Rahmen der Seminararbeit entwickelten Skripte in Python, die auf den Sachverhalt genau zugeschnitten sind.

Als Ablageart ist Hive zu empfehlen, da die Daten mit HQL abgefragt und ausgewertet werden können. Im Gegensatz dazu müssten bei der Ablage der Daten auf dem HDFS diese erst für die weitere Verarbeitung aus den Dateien extrahiert werden. Außerdem besteht bei HDFS das Small-Files-Problem, wodurch Hive im Vergleich den Speicher effizienter ausnutzt.

Somit scheint im Hinblick auf Skalierbarkeit, Performance und Weiterverarbeitbarkeit das Python-Skript zur Ablage der MES Hydra API-Daten in Hive am besten geeignet zu sein.

Im Rahmen der Bearbeitung der Seminararbeit traten folgende Probleme auf. Bevor mit dem HDP-Cluster gearbeitet werden konnte, mussten mehrere Neustarts des Servers und Anpassungen der Konfiguration vorgenommen werden. Beispielsweise gab es Probleme mit der Authentifizierung in Kerberos, das darauffolgend deaktiviert wurde und einige Pakete des Servers waren nicht auf dem aktuellen Stand.

Eine Weiterentwicklungsmöglichkeit besteht im Bereich der Automatisierung der Lösungen. Hierfür muss zuvor geklärt werden, wie oft neue Daten in der Digitalen Fabrik erzeugt werden. NiFi ist einfach automatisierbar, da den Prozessoren fixes Startzeiten und Intervalle mitgegeben werden können. Die Python-Lösung muss mittels des Cron-Daemon automatisiert werden, um die Ausführung als Dienst zu ermöglichen. Hier kann ebenfalls eine Festlegung der Startzeiten und Intervalle erfolgen.

Des Weiteren kann in einer Weiterentwicklung die Unterstützung des Abrufs von "list…"-Services mit obligatorischen Parametern aufgenommen werden.

Literaturverzeichnis

Apache HBase Team (2020)

Apache HBase Team (Hrsg.; 2020): Apache HBase Reference Guide. "https://HBase.apache.org/book.html". Abruf am 2020-05-26.

Apache Software Foundation (o. J. a)

Apache Software Foundation: (Hrsg.; o. J.): Apache Hive. "https://cwiki.apache.org/confluence/display/HIVE". Abruf am 2020-05-22.

Apache Software Foundation (o. J. b)

Apache Software Foundation (Hrsg.; o. J.): HDFS Architecture Guide. "https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html”. Abruf am 2020-05-20.

Apache NiFi (o. J.)

Apache NiFi (Hrsg.; o. J.): Apache NiFi. "https://NiFi.apache.org/". Abruf am 2020-05-21.

Auch et al. (2017)

AUCH, Alexander; HÄHRE, Stephan; KUHN, Christian (2017): Labor Fertigungs- und Informationsmanagement (Digitale Fabrik / Industrie 4.0). Präsentation der DHBW Mosbach, Kompetenzzentrum Fertigungs- und Informationsmanagement, Fakultät Technik.

Bracht et al. (2018)

BRACHT, Uwe; GECKLER, Dieter; WENZEL, Sigrid (2018): Digitale Fabrik. Methoden und Praxisbeispiele. 2. Auflage, Springer-Verlag GmbH Deutschland, Berlin.

Brauckmann (2019)

BRAUCKMANN, Otto (2019): Digitale Revolution in der industriellen Fertigung, 1. Auflage, Springer Vieweg, Springer-Verlag GmbH Deutschland, Berlin.

Blaint (2009)

BALINT, Szele (2009): The Small Files Problem. "https://blog.cloudera.com/the-small-files-problem/". Abruf am 2020-05-20.

Chang et al. (2006)

CHANG, Fay; DEAN, Jeffrey; GHEMAWAT, Sanjay; HSIEH, Wilson C.; WALLACH, Deborah A.; BURROWS, Mike; CHANDRA, Tushar; FIKES, Andrew; GRUBER, Robert E. (2006): Bigtable: A Distributed Storage System for Structured Data. "https://storage.googleapis.com/pub-tools-public-publication-data/pdf/68a74a85e1662fe02ff3967497f31fda7f32225c.pdf". Google, Inc. Abruf am 2020-05-26.

Cloudera, Inc. (o. J.)

Cloudera, Inc. (Hrsg.; o. J.): Apache Hive 3 architectural overview. "https://docs.cloudera.com/HDPDocuments/HDP3/HDP-3.1.5/hive-overview/content/hive-apache-hive-3-architectural-overview.html". Abruf am 2020-05-22.

DHBW Mosbach (o. J.)

DHBW Mosbach (Hrsg.; o. J.): Living Lab – Industrie 4.0. "https://www.mosbach.dhbw.de/forschung-transfer/kompetenzzentren/kompetenzzentrum-fertigungs-und-informationsmanagement/living-lab-industrie-40/". Abruf am 2020-05-05.

Du (2018)

DU, Dayong (2018): Apache Hive Essentials. 2. Auflage. Packt Publishing. Burmingham.

Fielding (2000)

FIELDING, Roy Thomas (2000): Architectural Styles and the Design of Network-based Software Architectures. "https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm". Abgerufen am 2020-05-26.

fraunhofer.de (2015)

Fraunhofer.de (Hrsg.; 2015): MES D.A.CH. Verband veröffentlicht UMCM-Spezifikation. "https://kommunikation.iosb.fraunhofer.de/servlet/is/100196/". Abruf am 2020-05-07.

Gerberich (2011)

GERBERICH, Thorsten (2011): Lean oder MES in der Automobilzulieferindustrie - Ein Vorgehensmodell zur fallspezifischen Auswahl. 1. Auflage. Gabler Verlag, Springer Fachmedien, Wiesbaden.

JSON.org (o. J.)

JSON.org (Hrsg.; o. J.): Einführung in JSON. "https://www.JSON.org/JSON-de.html". Abruf am 2020-05-04.

Kletti (2015)

KLETTI, Jürgen (2015): MES - Manufacturing Execution System. Moderne Informationstechnologie unterstützt die Wertschöpfung. 2. Auflage, Springer Verlag.

Kletti et al. (2019)

KLETTI, Jürgen; DEISENROTH, Rainer (2019): MES-Kompendium - Ein Leitfaden am Beispiel von HYDRA. 2. Auflage. Springer Vieweg, Springer-Verlag GmbH Deutschland.

Luber et al. (2016)

LUBER, Stefan; LITZEL, Nico (2016): Definition. Was ist Hadoop?. "https://www.bigdata-insider.de/was-ist-hadoop-a-587448/". Abruf am 2020-05-26.

Luber et al. (2017)

LUBER, Stefan; LITZEL Nico (2017): Definition: Was ist Hive?. "https://www.bigdata-insider.de/was-ist-hive-a-654184/". Abruf am 2020-05-22.

MPDV (o. J.)

MPDV (Hrsg.; o. J.): Manufacturing Execution System HYDRA. "https://www.mpdv.com/de/produkte-loesungen/mes-hydra/" Abruf am 2020-05-04.

MPDV Mikrolab GmbH (2018)

MPDV Mikrolab GmbH (Hrsg.; 2018): Handbuch Service Interface SCS-SIF 3.0. Version 1.0.14449.

Posey (2013)

POSEY, Brien (2013): Hadoop-Cluster: Vorteile und Herausforderungen für Big-Data-Analytik. "https://www.computerweekly.com/de/tipp/Hadoop-Cluster-Vorteile-und-Herausforderungen-fuer-Big-Data-Analytik". Abruf am 2020-05-22.

PyPA (2020)

PyPA (Hrsg.; 2020): Installing Packages. "https://packaging.python.org/tutorials/installing-packages/". Abruf am 2020-05-21.

Python Software Foundation (2020)

Python Software Foundation (Hrsg.; 2020): Find, install and publish Python packages with the Python Package Index. "https://pypi.org/". Abruf am 2020-05-21.

Rossy (2015)

ROSSY, Ngoukam Monkam Charly (2015): NoSQL Datenbank Technologie: HBase. "https://pi.informatik.uni-siegen.de/Mitarbeiter/mrindt/Lehre/Seminare/NoSQL/einreichungen/NOSQLTEC-2015_paper_14.pdf". Abgerufen am 2020-05-26.

Tutanch et al. (2017)

TUTANCH; LITZEL, Nico (2017): Definition. Was ist Hortonworks. "https://www.bigdata-insider.de/was-ist-hortonworks-a-623004/". Abruf am 2020-05-27.

VDI (2008)

VDI-Fachbereich Fabrikplanung und -betrieb (2008): VDI 4499 Blatt 1. Digitale Fabrik – Grundlagen. VDI-Gesellschaft Produktion und Logistik.

Anhang

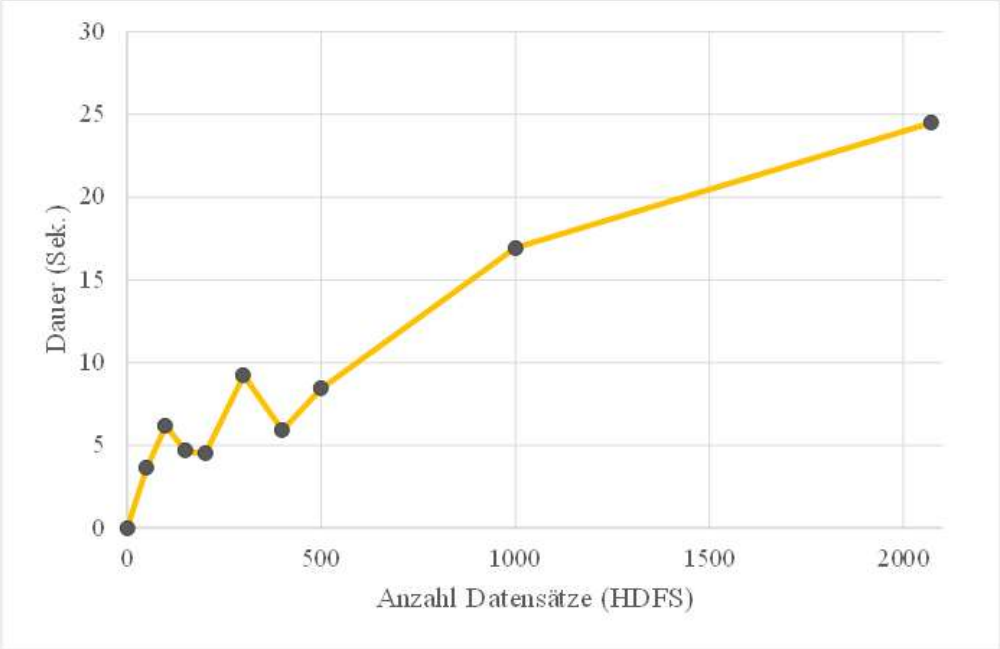
Anhang-Verzeichnis

A1 Performancetest Ablage im HDFS per NiFi-Routine – Aufschlüsselung der Dauern

HDFS (50)	HDFS (100)	HDFS (150)	HDFS (200)	HDFS (300)	HDFS (400)	HDFS (500)	HDFS (1000)	HDFS (alle)
2.350	2.670	2.187	2.335	2.265	2.453	2.669	3.115	2.715
0.582	1.283	0.371	0.843	0.414	0.624	0.566	0.604	0.865
0.034	0.032	0.076	0.187	1.197	0.266	0.367	1.204	1.975
0.251	0.214	0.520	0.314	1.502	0.721	0.968	2.008	3.781
0.047	0.063	0.418	0.126	1.292	0.375	0.520	1.153	1.963
0.279	0.418	0.644	0.432	1.765	0.938	1.630	3.669	7.008
0.025	0.044	0.047	0.076	0.353	0.112	0.280	0.803	0.925
0.002	0.017	0.003	0.002	0.021	0.011	0.025	0.070	0.049
0.089	1.437	0.390	0.172	0.426	0.422	1.406	4.326	5.210
SUMME	**3.659**	**6.178**	**4.656**	**4.487**	**9.235**	**5.922**	**8.431**	**16.952**	**24.491**

A2 Performancetest Ablage im HDFS per NiFi-Routine – Grafische Darstellung

Anzahl Datensätze (HDFS)	Dauer (Sek.)
0	0
50	3.659
100	6.178
150	4.656
200	4.487
300	9.235
400	5.922
500	8.431
1000	16.952
2071	24.491

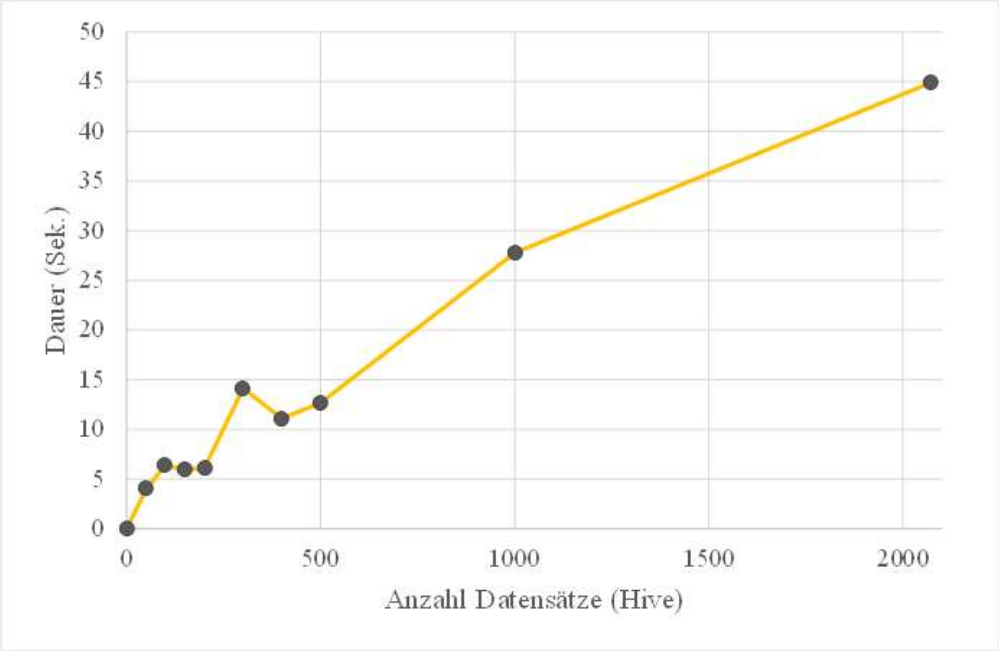


A3 Performancetest Ablage in Hive per NiFi-Routine – Aufschlüsselung der Dauern

Hive (50)	Hive (100)	Hive (150)	Hive (200)	Hive (300)	Hive (400)	Hive (500)	Hive (1000)	Hive (alle)
2.350	2.670	2.187	2.335	2.265	2.453	2.669	3.115	2.715
0.582	1.283	0.371	0.843	0.414	0.624	0.566	0.604	0.865
0.034	0.032	0.076	0.187	1.197	0.266	0.367	1.204	1.975
0.251	0.214	0.520	0.314	1.502	0.721	0.968	2.008	3.781
0.047	0.063	0.418	0.126	1.292	0.375	0.520	1.153	1.963
0.087	0.411	0.653	0.452	1.466	1.007	1.603	3.600	6.558
0.278	0.383	0.616	0.319	1.810	1.052	1.562	3.611	6.010
0.014	0.026	0.039	0.189	0.138	0.133	0.180	0.773	0.872
0.379	1.308	1.075	1.439	4.058	4.496	4.239	11.749	20.140
SUMME	**6.390**	**5.955**	**6.204**	**14.142**	**11.127**	**12.674**	**27.817**	**44.879**

A4 Performancetest Ablage in Hive per NiFi-Routine – Grafische Darstellung

Anzahl Datensätze (Hive)	Dauer (Sek.)
0	0
50	4.022
100	6.390
150	5.955
200	6.204
300	14.142
400	11.127
500	12.674
1000	27.817
2071	44.879



Ehrenwörtliche Erklärung

Hiermit erklären wir ehrenwörtlich,

(1) dass wir die vorliegende Arbeit mit dem Titel

Speicherung von MES-Daten (MPDV-Hydra) der "Digitalen Fabrik" in dem HDP (Hortonworks Data Platform)-Cluster

ohne fremde Hilfe angefertigt haben,

(2) dass wir alle wörtlich oder sinngemäß übernommenen Zitate aus anderen Quellen an den entsprechenden Stellen innerhalb der Arbeit eindeutig gekennzeichnet haben,

(3) dass diese Arbeit noch nicht in gleicher oder ähnlicher Form sowie vollständig oder auszugsweise einer anderen Prüfungsbehörde vorgelegt wurde,

(4) dass alle von uns eingereichten Versionen in Papierform inhaltlich absolut identisch sind,

(5) dass alle von uns eingereichten digitalen Versionen der Arbeit inhaltlich zu 100 Prozent mit den ausgedruckten und eingereichten Versionen in Papierform übereinstimmen.

Wir sind uns bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Ort, Datum	Christian Bonfert
Ort, Datum	Katharina Ehrmann
Ort, Datum	Maria Fladung
Ort, Datum	Lukas Habermann
Ort, Datum	Roman Nolde
Ort, Datum	Tom Schmelzer

Link zum GitHub-Repository

Das GitHub-Repository mit allen Python-Skripten und sonstigen Unterlagen ist zu finden unter:

https://github.com/chribon/DHBW-Integrationsseminar

Unterteilt ist das Repository in vier Ordner:

- **API_Inspection**
  Dieser Ordner enthält ein Jupyter Notebook, das während der Sichtung der MES Hydra API erstellt wurde und in dem z. B. die Anzahlen der Domains ermittelt wurden und aufgelistet sind. Zudem sind die einzelnen Übersichten von Domains und Services im JSON-Format abgelegt.
- **HDFS**
  Abgelegt sind hier das für den Performancetest genutzte Python-Skript und ein Skript, das alle API-Endpunkte abfragt. Zusätzlich ist eine Auflistung aller zu installierenden Python-Module vorhanden.
- **Hive**
  Neben einer Datei mit den zu installierenden Python-Modulen ist das Python-Skript zur Abfrage und Ablage aller API-Endpunkte enthalten und das für die Performancetests verwendete Skript. Hierbei handelt es sich je um Variante drei, die die Zwischenspeicherung im HDFS nutzt. Die unter Kapitel 4.4 Bereich Python Eigenentwicklung beschriebenen Varianten eins und zwei sind im Unterordner 'variants1and2' abgelegt.
- **Dokumentation**
  Hier ist das ausgearbeitete Dokument zu finden.

Die Python-Skripte zur Ablage auf HDFS und in Hive sind zusätzlich in einer virtuellen Python-Umgebung auf dem HDP-Cluster-Server zu finden (Order des Benutzers wi17mes → Ordner hdp/src).