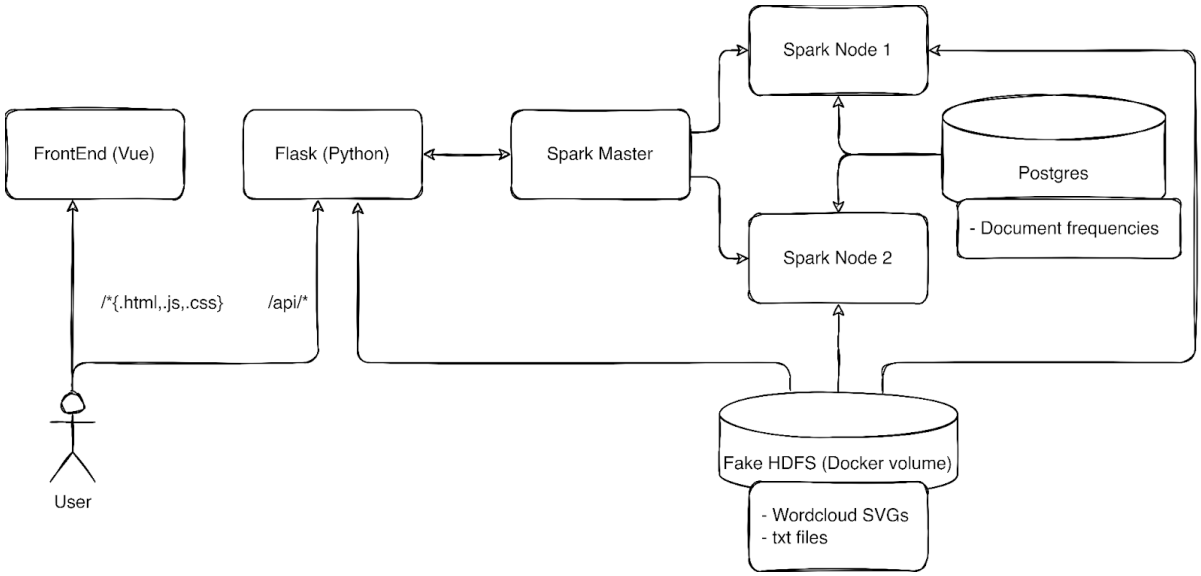
Dokumentation

CA2 - Wordcloud Berechnung

Komponenten:

FrontEnd:	VueJS, Javascript
Webserver:	Flask, Python
Analyse-System:	Spark, Python
Datenbank:	PostgresSQL

Architektur:



Ablauf:

File-Upload:

1. User lädt Text-Datei über das Front-End auf den Flask-Server hoch
2. Flask speichert die Textdatei ab und triggert beim Front-End einen Pagereload
3. Front-End lädt neu und fragt den aktuellen Bestand aller Text-Dateien, sowie die dazugehörigen Wordclouds als SVG Bilder ab
4. Flask versucht die Wordcloud-SVG-Datei zu den Namen der Textdatei zurückzugeben
5. Falls das Wordcloud-Bild nicht vorhanden ist:
    a. Flask triggert über den Spark-Master die Erstellung des fehlenden Wordcloud-Bildes

b. Flask locked den Api-Endpoint, um parallele Aufrufe zu stoppen, sodass weitere Aufrufe warten bis das fertige Wordcloud-Bild existiert und diese dann direkt zurückliefern
c. Spark Master macht Magie mit seinen 2 Helferlein (Nodes)
d. Die Spark Nodes laden die Textdatei und bestimmen für diese die TFiDF mit Hilfe der DF Werte aus der Postgres Datenbank. Fehlende DF Werte werden nach dem Join auf 1 gesetzt
e. Die Spark Nodes erstellen mit PySpark und dem Wordcloud Package eine SVG Datei, welche im fake HDFS (einfaches Docker Volume) abgelegt wird
f. Flask gibt den Lock des Api-Endpoints wieder frei
g. Flask gibt das fertig erstellte Bild an alle wartenden Aufrufer des Api-Endpoints zurück

Batch:

1. Nutzer drückt den Batch-Knopf
2. Front-End schickt Flask einen HTTP-Request
3. Flask startet den Batch-Job über das Spark System
    a. Spark-Nodes laden alle Textdateien und ermitteln die DF
    b. Spark-Nodes schreiben die DF in die Postgres-Datenbank
    c. Flask löscht nach dem Zählen die Wordcloud-Bilder
    d. Flask sendet “success”-Antwort an das wartenden Front-End
    e. Front-End führt einen Pagereload durch
    f. Front-End lädt durch den Pagereload alle Bilder neue und lässt somit alle Wordcloud-Bilder neu generieren, da diese im vorherigen Schritt von Flask gelöscht wurden (siehe File-Upload, Punkt 3)

Erfahrungen:

- Static Vue-Frontend über Flask zu deployen war schwer => static_folder in Flask Konstruktor setzen für Single-Page-Applications
- Anfangs kein Zugriff auf den Server und die DB, von außerhalb des Dockercontainers => Ports wurden nicht forwarded und listen-host war nicht auf "0.0.0.0" gesetzt
- TFiDF muss mit log10 und math.ceil normiert werden, um diese über die Python Wordcloud Library anzeigen zu können
- Das Verwenden eines Docker Volumes zum Austauschen von Dateien zwischen den Spark-Nodes um dem Flask-Backend als eine Art HDFS erlaubt eine sehr einfache Entwicklung
- Die Zusammenführung mehreren Container durch verschiedenen Services und Netzwerklevel kann ziemlich komplex werden und durch die verschiedenen Ebenen (Host-Shell, Shell im Docker Container, …) schwer durchschaubar sein