Data Analytics Project bei Datev

## Kurze Projektbeschreibung

Die Datev steht vor der Herausforderung, dass viele verschiedenen Metriken über verschiedene Wege in den Appdynamics Plattform gelangen. Der Controller speichert diese Metriken in seiner Datenbank und macht Sie über den Metrik Browser, die UI und per API verfügbar. In einigen Anwendungsfällen – wie bei Applikationsübergreifenden Dashboards – sind diese Daten leider nicht über eine dynamische Query (wie für Analytics Data) abzufragen.

Um diese Daten dynamischer abzurufen, sollen bestimmte Metriken aus dem Controller innerhalb des Event-Service verfügbar gemacht werden, damit die Datev diese dann über Appdynamics Dynamic Query Language abfragen kann.

Um diese Daten auch per Query abfragbar zu machen ist es von notwendig, diese Daten per API aus dem Controller abzufragen und in den Event-Service zu pushen. Dazu kann die Controller Metrics API (<https://docs.appdynamics.com/display/PRO45/Metric+and+Snapshot+API>) sowie die Analytics Events API (<https://docs.appdynamics.com/display/PRO45/Analytics+Events+API>) genutzt werden.

## Ressourcenbeschränkungen

Die Anzahl der zu verarbeiteten Metriken ist durch die zur Verfügung stehenden Ressourcen, die für das Skript zur Verfügung stehen beschränkt. Darüber hinaus sind die API-Aufrufe auch durch die verfügbaren Ressourcen auf der AppDynamics Platform beschränkt. Das kann verschiedene Auswirkungen auf das Gesamt-System haben:

1. Falls die Ressourcen auf dem Server des Synchronisierungsskriptes zu knapp sind, kann sich eine beliebig große Verzögerung in den zu synchronisierenden Daten aufbauen. Dadurch stehen die aktuellen Daten aus dem Controller nur verzögert auf dem Event-Service zur Verfügung.
2. Falls die API-Aufrufe auf dem Controller oder dem Event-Service zu viel Last erzeugen, kann dadurch deren Perfomance insgesamt runter gehen.

Um mit den vorhandenen Ressourcen optimal nutzen zu können setzt das Synchronisierungsskript auf einen **ThreadPool** um den Overhead zu minimieren und auf hohe Parallelisierung zu setzen. Details dazu finden sich unter https://docs.python.org/3/library/concurrent.futures.html

## Umsetzung über die Analytics Event Service API

Um die Daten von den jeweiligen lokalen Skripten zur Appdynamics Plattform zu übermitteln, verwenden wir die Analytics Event Service API. Diese API wird in der Dokumentation umfassend beschrieben: <https://docs.appdynamics.com/display/PRO45/Analytics+Events+API>

Das Skript kann sich mit Hilfe eines API Keys der im Headerfeld **X-Events-API-Key** bezeichnet ist gegenüber dem Eventservice autorisieren. Das Anlegen eines solchen Schlüssels ist hier beschrieben: <https://docs.appdynamics.com/display/PRO45/Managing+API+Keys>

Die folgenden Datenstruktur nutzen wir um die abgefragten Daten im Event-Service zu hinterlegen.

1. metricName (String) - Name der selbst vergeben werden kann innerhalb der **metrics.txt**
2. application (String) – Applikationsname aus welcher die Metrik stammt
3. metricpath (String) – Vollständiger Metrik-Pfad
4. metricpath1 (String), .., metricPath11 (String) – Pfad aufgeteilt am Trenner „|“
5. startTimeInMillis (Date) – Zeitstempel wann der Wert vom Controller erhoben wurde
6. value (Integer) – Wert des ermittelnden Wertes

## Technische Ressourcen

Um die beschriebenen Anforderungen zu erfüllen werden die folgenden 3 Skripte zur Verfügung gestellt

1. controller-to-event-service-syncher.py
   1. Muss stetig laufen um die Controller-Daten zu dem Event-Service zu übertragen.
2. manage\_schema.py
   1. Kann genutzt werden, um neue Schemas im Event-Service anzulegen oder zu löschen. Die Struktur ist fest vorgegeben.
3. query-deleter.py
   1. Kann genutzt werden, um existierende Daten zu löschen.

Alle 3 Skripte sind auf die Konfigurationsdatei **config.txt** angewiesen. Die entsprechenden Parameter sind direkt in der config.txt beschrieben.

## Ablauf Beschreibung

**controller-to-event-service-syncher.py**

1. Einlesen der Start-Parameter und der config.txt
2. Autorisierung beim Controller und dem Event-Service
3. Einlesen der metrics.txt
4. Wenn start\_time und end\_time gesetzt sind:
   1. Prozessiere den Zeitraum zwischen start\_time und end\_time mit der Methode process\_all\_metrics
5. Sonst:
   * 1. Lege einen Schedule an, der alle N Sekunden (wie in der config.txt im Wert schedule\_time gesetzt) die Synchronisierung mit dem Controller aufruft.

**manage\_schema.py**

1. Einlesen der Start-Parameter und der config.txt
2. Autorisierung beim Event-Service
3. Wenn operation = „create“: Aufruf von Event-Service-API mit **POST** **/events/schema/<SchemaName>** mit Schema-Struktur
4. Wenn operation = „delete“: Aufruf von Event-Service-API mit **DELETE** **/events/schema/<SchemaName>**

**Query\_deleter.py**

1. Einlesen der Start-Parameter und der config.txt
2. Autorisierung beim Event-Service
3. Abfrage der übergebenden Query
4. Bestätigung der Zeilen die zu löschen sind durch den Benutzer
5. Wenn die Zeilen korrekt bestätigt wurden:
   1. Erzeuge Autorisierungscode per Base64 mit (Global Account Name & Controller Access Key)
   2. Aufruf der Event-Service-API **/events/query/\_delete** mit der Query als Parameter