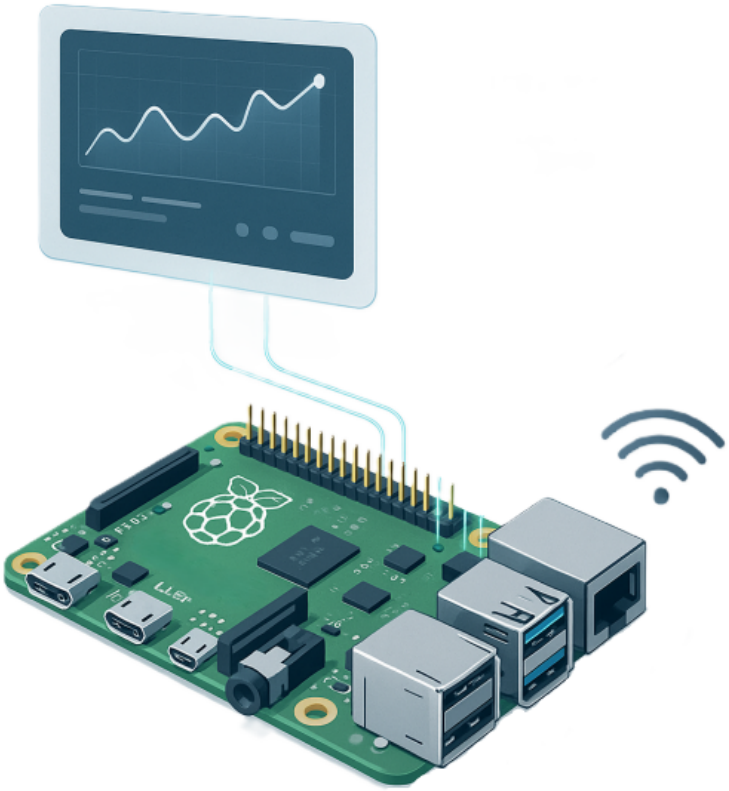
Projektarbeit I: SBB Live Monitor

Verteilte Systeme II - Hands-on



Gruppe: Theophile Becker / Theo Metzger
Datum: 7. Dezember 2025

Inhaltsverzeichnis

1 Installation und Konfiguration **2**
1.1 Schritte und Auswirkungen 2

2 Dashboard Zugriff **2**

3 Visualisierung und Datenherkunft **2**
3.1 Datenquelle 2
3.2 Plots 2

4 Verwaltung der Dienste (Start/Stop) **3**
4.1 Webserver (Systemd) 3
4.2 Datensammler (Cronjob) 3

5 Zugangsdaten VM **4**

6 Aufgabenzuteilung **4**

1 Installation und Konfiguration

Folgende Schritte wurden auf einer frischen Ubuntu 25.10 VM unternommen, um das System lauffähig zu machen.

1.1 Schritte und Auswirkungen

- **System-Update & Abhängigkeiten:** Zunächst wurden die Paketquellen aktualisiert und notwendige Systemwerkzeuge installiert.

  ```
  sudo apt update && sudo apt install python3-pip python3-venv sqlite3
  ```

  Auswirkung: Das System ist aktuell und verfügt über die Python-Runtime sowie die SQLite-Datenbank-Engine.

- **Projektumgebung (Virtual Environment):** Ein isoliertes Environment (`venv`) wurde erstellt und die Bibliotheken `flask` und `requests` installiert. *Auswirkung:* Projektabhängigkeiten sind sauber vom globalen System getrennt.

- **Datenbank-Initialisierung (SQLite):** Das Python-Script erstellt beim ersten Start automatisch die Datei `sbb.db` mit einem relationalen Schema (Tabellen: `stations` und `departures`). *Auswirkung:* Persistente Speicherung der Stammdaten und Bewegungsdaten.

- **Automatisierung (Cronjob):** Ein Cronjob wurde eingerichtet, der das Fetch-Script jede Minute ausführt. *Auswirkung:* Automatische Anreicherung historischer Daten im Hintergrund.

- **Dienst-Einrichtung (Systemd):** Ein Service-File (`sbb-dashboard.service`) wurde erstellt. *Auswirkung:* Der Webserver startet automatisch beim Booten der VM (Always-on).

2 Dashboard Zugriff

Das Dashboard wird über den Browser innerhalb der VM aufgerufen.

- **URL:** http://localhost:5000

Das Frontend besitzt einen `meta-refresh` Tag, wodurch sich die Seite alle 60 Sekunden automatisch aktualisiert, um die neuesten Daten aus der SQLite-Datenbank anzuzeigen.

3 Visualisierung und Datenherkunft

3.1 Datenquelle

Die Daten werden über die öffentliche API **transport.opendata.ch** bezogen. Das Skript fragt spezifisch die Abfahrten des Bahnhofs **Zürich HB** ab.

3.2 Plots

Das Dashboard visualisiert die lokal gespeicherte Historie:

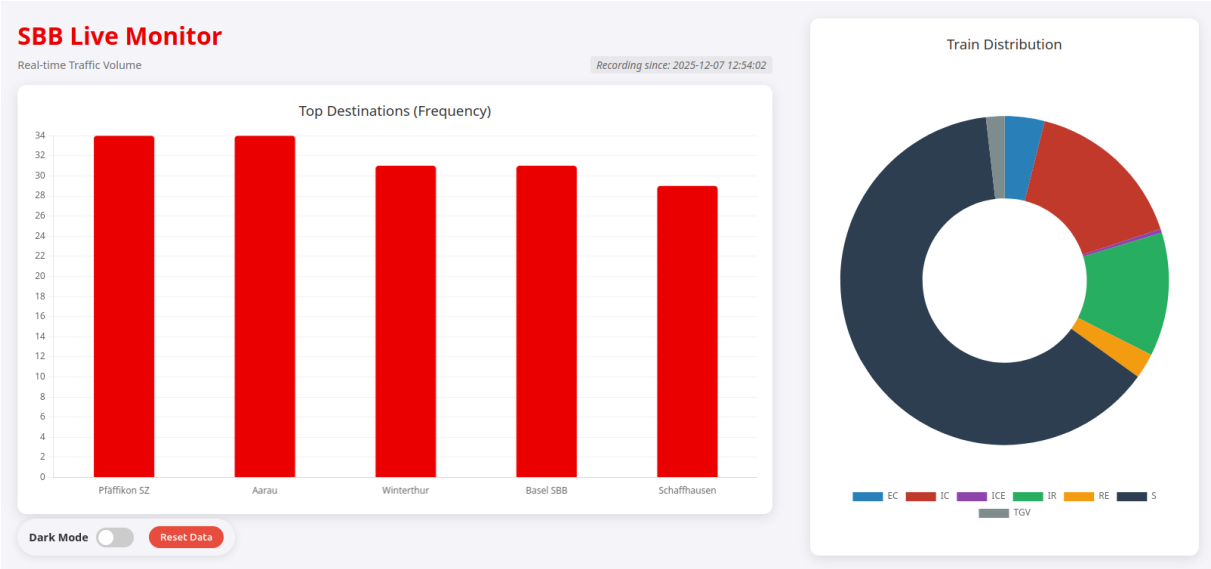


Abbildung 1: Ansicht des SBB Live Monitors im Browser

1. **Linker Plot (Balkendiagramm): Top Destinations**
   Zeigt die 5 häufigsten Zielorte basierend auf der Frequenz (Anzahl der Züge). Dies gibt Aufschluss über das Verkehrsaufkommen und die Hauptrichtungen.

2. **Rechter Plot (Donut-Chart): Train Distribution**
   Zeigt die Verteilung der Zugkategorien (S-Bahn, IR, IC, etc.). Dies charakterisiert den Bahnhofstyp (z.B. Regionalverkehr vs. Fernverkehr).

4 Verwaltung der Dienste (Start/Stop)

Das System besteht aus zwei unabhängigen Komponenten:

4.1 Webserver (Systemd)

Der Dashboard-Server wird über `systemctl` gesteuert:

```
# Status pruefen
sudo systemctl status sbb-dashboard

# Stoppen / Starten / Neustarten
sudo systemctl stop sbb-dashboard
sudo systemctl start sbb-dashboard
sudo systemctl restart sbb-dashboard
```

4.2 Datensammler (Cronjob)

Das Sammeln der Daten erfolgt im Hintergrund:

```
# Bearbeiten
crontab -e

# Logs pruefen
cat ~/sbb_project/cron.log
```

5 Zugangsdaten VM

- **Benutzername:** beckert
- **Passwort:** 1234
- **Sudo-Rechte:** Vorhanden

6 Aufgabenzuteilung

- **Théophile:** Gesamte Umsetzung des Projekts.
  - Aufsetzen der Ubuntu VM und Installation der Pakete.
  - Programmierung des Python ETL-Scripts (Extract-Transform-Load).
  - Implementierung der SQLite-Datenbanklogik.
- **Theo:** Gesamte Umsetzung des Projekts.
  - Entwicklung des Flask Webservers und des Frontends (Chart.js, Dark Mode).
  - Einrichtung der Automatisierung (Cron & Systemd).