Technische Dokumentation

Energiebasierte Terminplanung

Philipp Arbogast (2351987), Christina Cser (2212074), Nils Heilemann (2278387), Louis Mauser (2268421), Hannes Metz (2340041)

Projektseminar

Lehrstuhl für BWL und Wirtschaftsinformatik
Julius-Maximilians-Universität Würzburg

Betreuer: Prof. Dr. Axel Winkelmann

Bearbeitungszeitraum: 01.11.2022 bis 31.03.2023

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Konfiguration des virtuellen Servers

Dieses Kapitel beinhaltet das Aufsetzen des virtuellen Servers und die Bereitstellung der grundlegenden Komponenten. Abschließend wird gezeigt, in welchem Verzeichnis der Source-Code der Applikation abgelegt wird.

1.1 Aufsetzen des virtuellen Servers

Der virtuelle Server zur Entwicklung des Portals zur energiebasierten Terminplanung ist bei dem Anbieter „MyVirtualServer“ (https://www.myvirtualserver.com/de) gehosted. Dort wurde die Konfiguration „**Cloud Virtual Server – NVMe SSD PICO“** ausgewählt. Diese beinhaltet die folgenden Komponenten:

- CPU-Kerne: 1x (2,90 GHz)
- RAM 2 GB
- Speicherplatz 20 GB SSD
- Bandbreite 250 Mbit/s
- **Betriebssystem: Ubuntu 22.04 LTS**

> ***Disclaimer**: Alle Befehle werden in der Dokumentation mit root-Rechten ausgeführt. In einer anderen Konfiguration, bzw. mit einem anderen Account müssen Befehle eventuell mit „sudo“ ausgeführt werden. Das Team des Projektseminars kann nicht garantieren, dass die bereitgestellte Applikation bei der Nutzung anderer Softwarekomponenten / einer anderen Linux-Distribution mit abgeänderten Einstellungen die vollständige Funktionalität bereitstellt.*

1.2 Installation des LAMP-Stacks

Der LAMP-Stack setzt sich aus den Komponenten **L**inux, **A**pache2, **M**ySQL und **P**HP zusammen. Diese Komponenten (vor allem Apache2 und MySQL) sind Grundvoraussetzungen für die korrekte Funktionalität der Applikation. Durch die folgenden Befehle lassen sich alle benötigten Komponenten installieren:

```
apt update
apt install apache2 -y
apt install mysql-server -y
apt install php libapache2-mod-php php-mysql -y
```

***Disclaimer**: Eventuell muss für die Apache Konfiguration in der Firewall der Port freigeschalten werden. Dies ist bei der Beispielkonfiguration nicht der Fall.*

1.3 Source Code hochladen

Im Verzeichnis /var/www/PJS/ wird der Source Code hochgeladen. Zu Entwicklungszwecken werden jedem Benutzer volle Lese- und Schreibberechtigungen für das Verzeichnis bereitgestellt. Für den produktiven Betrieb ist es ausreichend, wenn der Apache2-Nutzer (Nativ ist das der Nutzer „www-data“) volle Berechtigungen für die Ordnerstruktur hat.

```
cd /var/www/
git clone https://github.com/jmu-informationsystemsgroup/pjs-ebd-
ws22_23.git PJS/
cd PJS
git checkout dev
cd ..
chmod 777 -R PJS
```

Weiterhin muss die Python3-Entwicklungsumgebung und mod_wsgi für den Apache2-Webserver installiert werden.

```
apt-get install libapache2-mod-wsgi-py3 python-dev-is-python3
a2enmod wsgi
```

Abschließend werden in der Einrichtung noch alle benötigten PIP-Pakete für das Ausführen der Flask-Webanwendung installiert. Zu diesem Zweck wird eine *requirements.txt* Datei im Ordnerverzeichnis bereitgestellt.

```
apt install python3-pip
cd /var/www/PJS
pip install -r requirements.txt
```

2 Konfigurieren und Aktivieren eines virtuellen Hosts

Der Apache2-Webserver nutzt sogenannte „virtuelle Hosts“, um Inhalte in gewissen Ports zur Verfügung zu stellen. Für die Flask-Applikation wird ein neuer virtueller Host erstellt.

> ***Disclaimer:*** *In der gesamten Dokumentation wird „nano“ als Texteditor verwendet. Hier können auch andere Standardeditoren wie Sublime oder VIM verwendet werden.*

```
cd /etc/apache2/sites-available/
nano PJS.conf
```

Folgender Inhalt wird in die neu erstellte Datei PJS.conf geschrieben:

```
<VirtualHost *:80>
        TimeOut 300
        ServerName localhost
        ServerAdmin webmaster@localhost
        DocumentRoot /var/www/PJS

        WSGIDaemonProcess yoursite threads=5 python-path=/var/www/PJS
        WSGIScriptAlias / /var/www/PJS/config.wsgi
        WSGIPassAuthorization On

        <Directory /var/www/PJS>
                WSGIProcessGroup yoursite
                WSGIApplicationGroup %{GLOBAL}
                Require all granted
        </Directory>

        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Abschließend muss die Hosts-Datei noch aktiviert werden:

```
a2ensite PJS.conf
a2dissite 000-default.conf
systemctl reload apache2
```

Zusätzlich zu dem Port 80 kann mit einem SSL-Zertifikat auch der Port 443 mit SSL-Zertifikat genutzt werden. Da zu diesem Zweck eine kostenpflichtige Domain notwendig ist, wird in der Abgabe auf die Nutzung eines SSL-Zertifikats verzichtet. Eine Anleitung zum Verschlüsseln der Kommunikation eines Apache2 Webservers auf Ubuntu, welche auch im Zuge des Projektseminars genutzt worden ist, ist bei DigitalOcean zu finden: https://www.digitalocean.com/community/tutorials/how-to-secure-apache-with-let-s-encrypt-on-ubuntu-18-04 (Stand: 26.03.2023)

3 Einrichten interner Komponenten der Applikation

Zum Abrufen der vollen Funktionalität der Applikation muss eine MySQL-Datenbank erstellt werden und eine gültige Gurobi-Lizenz hinterlegt werden. Diese beiden Schritte werden nachfolgend in diesem Kapitel näher erörtert.

3.1 Aufsetzen der Datenbank

Zuerst muss MySQL geöffnet werden:

```
mysql
```

Durch das Eingeben folgender Befehle kann eine neue Datenbank namens „pjs“ und ein Nutzer mit vollen Zugriffsrechten erstellt werden:

```
CREATE DATABASE pjs;
USE pjs;
CREATE USER 'energy'@'localhost' IDENTIFIED BY 'PJS2022';
GRANT ALL PRIVILEGES ON * . * TO 'energy'@'localhost';
FLUSH PRIVILEGES;
Exit;
```

Abschließend kann eine Datenbankmigration der Flask-Applikation durchgeführt werden, damit die benötigten Tabellen automatisiert über das Paket „Flask-Migrate“ erstellt werden:

```
cd /var/www/PJS/
flask db init
flask db migrate -m "Initialer Commit"
flask db upgrade
```

3.2 Aktivieren der Gurobi-Lizenz

Bevor Gurobi genutzt werden kann, wird eine Lizenz dafür benötigt. Diese muss zuerst auf der Internetseite von Gurobi beschafft werden und danach aktiviert werden.

> ***Disclaimer**: Aufgrund des Lizenzmodells von Gurobi kann das Projektseminarteam keinen gültigen Zugangsschlüssel für Gurobi der Abgabe hinzufügen, da dieser nur für einen einzelnen Server erstellt werden kann. Die nachfolgenden Punkte beschreiben, wie eine Lizenz generiert werden kann und wie diese installiert wird.*

3.2.1 Erhalten einer Gurobi-Lizenz

In diesem Projekt wird eine akademische Gurobi-Lizenz genutzt. Zum Erhalt einer Gurobi-Lizenz muss ein Nutzerkonto auf https://www.gurobi.com und für die akademische Nutzung freigeschaltet werden. Zur Nutzung der Lizenz auf einem Webserver ist eine Web Server License (WLS) erforderlich. Diese kann über den WLS Manager (siehe Abbildung 1) erstellt werden. Für eine aktivierte Lizenz kann unter dem Menüpunkt „API Key“ ein API-Schlüssel generiert werden. Nach Erstellung eines neuen API-Schlüssels kann eine .lic-Datei heruntergeladen werden. Diese Lizenzdatei enthält alle relevante Lizenzdaten. Diese Daten müssen an bestimmter Stelle auf dem Webserver gespeichert werden. In dem nachfolgenden Schritt wird dieses Vorgehen beschrieben.

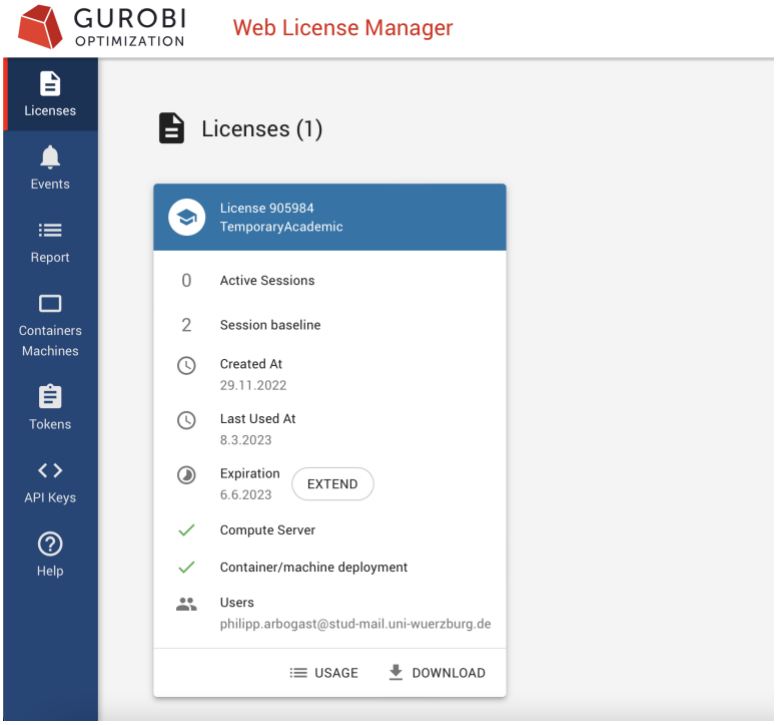

Abbildung 1: Gurobi Web License Manager

3.2.2 Einrichten von Gurobi

Gurobi muss in der Variante x64 Linux von der offiziellen Seite heruntergeladen (https://www.gurobi.com/downloads/gurobi-software/) und mittels SFTP auf den Applikationsserver verschoben werden. Im Projektseminar wird der Gurobi Optimizer in der Version 10.0.0 genutzt. Das tar.gz-Archiv kann dort entpackt und weiterverarbeitet werden. In dem folgenden Beispiel wird das tar.gz-Archiv in dem Ordner /home/ abgelegt.

```
cd /home/
tar -xzf gurobi10.0.0_linux64.tar.gz
cd gurobi1000/linux64/
python3 setup.py install
```

Weiterhin müssen folgende Einstellungen in den Systemvariablen hinterlegt werden. Für den Nutzer www-data (dem Linux-Account, der die Flask-Applikation ausführt) sind diese unter folgendem Ablageort änderbar:

```
nano /etc/apache2/envvars
```

Folgende Einstellungen müssen ergänzt werden:

```
export GUROBI_HOME=/home/gurobi1000/linux64/
export PATH=$GUROBI_HOME/bin:"$PATH"
export LD_LIBRARY_PATH=$GUROBI_HOME/lib:"$LD_LIBRARY_PATH"
export GRB_LICENSE_FILE=/home/gurobi1000/linux64/gurobi.lic
```

Die Variable GUROBI_HOME ist dabei der Pfad, unter dem Gurobi abgelegt worden ist. Zudem muss eine Lizenzdatei namens gurobi.lic im GUROBI_HOME-Pfad erstellt und befüllt werden. Den Prozess zum Erhalten einer Lizenz wird Kapitel 3.2.1 beschrieben. In der Datei sind die Parameter WLSACCESSID, WLSSECRET und LICENSEID hinterlegt.

4 Einrichten externer Komponenten der Applikation

Neben den internen Komponenten werden mit OpenWeatherMap und der Microsoft-365 Anbindung auch externe Komponenten benötigt. Den Prozess zum Hinterlegen der benötigten Parameter wird nachfolgend in diesem Kapitel beschrieben.

4.1 Anbindung von OpenWeatherMap

Für die Anbindung von OpenWeatherMap wird ein professioneller Account bei OpenWeatherMap benötigt.

> ***Disclaimer**: In der settings.cfg des Source Codes ist bereits ein gültiger API-Key enthalten (Stand 11.03.2023). Wir übernehmen keine Verantwortung, wie lange dieser API-Key nach der Abgabe seitens des kostenlosen Studentenprogramms von OpenWeatherMap gültig bleibt. In der Dokumentation ist daher beschrieben, wie das Betreuerteam einen neuen API-Key generieren kann.*

Auf der Seite https://openweathermap.org/price#current kann ein Abonnement ausgewählt werden. Das Projektseminarteam hat hier den „Developer Plan“ gewählt, da dieser für Studenten kostenlos ist. Nach der Registrierung eines Accounts kann der API-Schlüssel aus dem neu angelegten Benutzerkonto kopiert werden und über die Benutzeroberfläche in den Settings (http://your-tenant.domain/settings) in die Flask-Applikation eingepflegt werden.

> ***Wichtig**: Hier muss ein professioneller Account erstellt werden, da die Applikation nicht den Endpunkt api.openweathermap.org anspricht, sondern pro.openweathermap.org. Die kostenfreie Option ist daher nicht möglich.*

4.2 Anbindung eines Microsoft 365 Mandanten

> ***Disclaimer**: Für diesen Schritt wird ein M365-Tenant benötigt. In der Abgabe des Projektseminars werden bereits in der graph_settings.json gültige Werte für ClientID, TenantID und ClientSecret angegeben, die vom Betreuerteam genutzt werden können. Da es sich bei diesen IDs um einen Microsoft Developer Tenant handelt, übernehmen wir keine Verantwortung,*

wie lange diese Credentials funktionieren, da Microsoft den Zugang zu dem Developer-Tenant einstellt, sobald (laut Nutzungsbedingungen) die 90-tägige Nutzungsdauer vorbei ist und keine Entwickleraktivität festgestellt werden kann. Beispielsweise kann über folgende Methoden ein neuer, kostenloser M365-Demomandant angelegt werden:

- Microsoft Developer Program (vom Projektseminarteam empfohlen)
- Microsoft CDX Tenant (nur für Partnerunternehmen von Microsoft zugänglich)

Sobald ein Mandant bereitgestellt ist, kann dieser folgendermaßen konfiguriert werden:

4.2.1 Bearbeiten bestehender Nutzer

Um einen reibungslosen Betrieb zu gewährleisten, muss für jede der drei Maschinentypen ein Nutzer angelegt werden. Zu Testzwecken ist es ausreichend, lediglich bereits bestehende Nutzer (im Falle eines Developer Mandanten werden 16 Demo-Nutzer angelegt, welche bearbeitet werden können) anzupassen. Drei Nutzer müssen folgende Daten enthalten (Anpassbar mit Klick auf den Namen und Kontaktinformationen verwalten):

Vorname	**Anzeigename**	**Position**
5	Lötbad 5	pjs_machine
3x4	Lötbad 3/4	pjs_machine
welle	Wellenlöt	pjs_machine

Tabelle 1: Konfiguration der benötigten Nutzer im M365-Mandanten

4.2.2 Azure-App registrieren

Auf dem Azure Portal (https://portal.azure.com/) kann der Administrator des Mandanten eine neue App-Registrierung durchführen. In der Maske zur Registrierung einer Anwendung muss bei „Unterstützte Kontotypen“ die Option „Nur Konten in diesem Organisationsverzeichnis“ ausgewählt werden.

Der registrierten Applikation müssen nun Berechtigungen erteilt werden. Dazu wird der Menüpunkt API-Berechtigungen gewählt. Die **Anwendungsberechtigungen** (wichtig: hier ist auch die Auswahl von delegierten Berechtigungen möglich, das ist allerdings nicht zielführend) Calendars.Read, Calenadars.ReadWrite und User.Read.All müssen gesetzt werden und vom Admin freigegeben werden. Die Konfiguration ist der Abbildung 2 zu entnehmen.

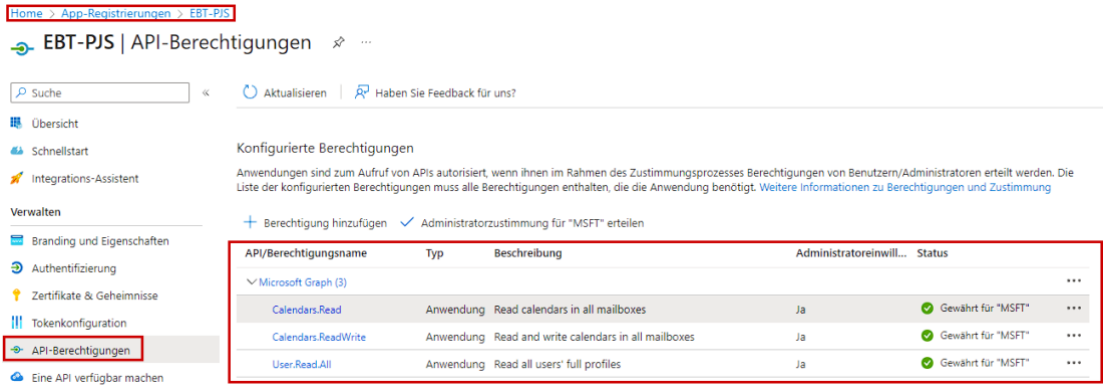

Abbildung 2: Konfiguration der Azure Applikation

Anschließend muss unter dem Menüpunkt „Zertifikate und Geheimnisse“ noch ein neuer Clientschlüssel generiert werden. Die Gültigkeit von diesem kann beliebig gesetzt werden. Der Inhalt der Spalte „Wert“ des neu generierten Schlüssels muss direkt kopiert werden, da dieser nur einmalig angezeigt wird (siehe Abbildung 3). Für die Flask-Applikation wird der Schlüssel in der Datei graph_settings.json im Eintrag **secret** hinterlegt.

Abbildung 3: Neues Client Secret anlegen

Abschließend wird der Menüpunkt Übersicht aufgerufen und hier der Wert Anwendungs-ID (Client) in graph_settings.json an der Stelle **client** geschrieben. Die Verzeichnis-ID (Mandant) wird an die Stelle **tenant** geschrieben. Die zu extrahierenden Werte sind Abbildung 4 zu entnehmen.

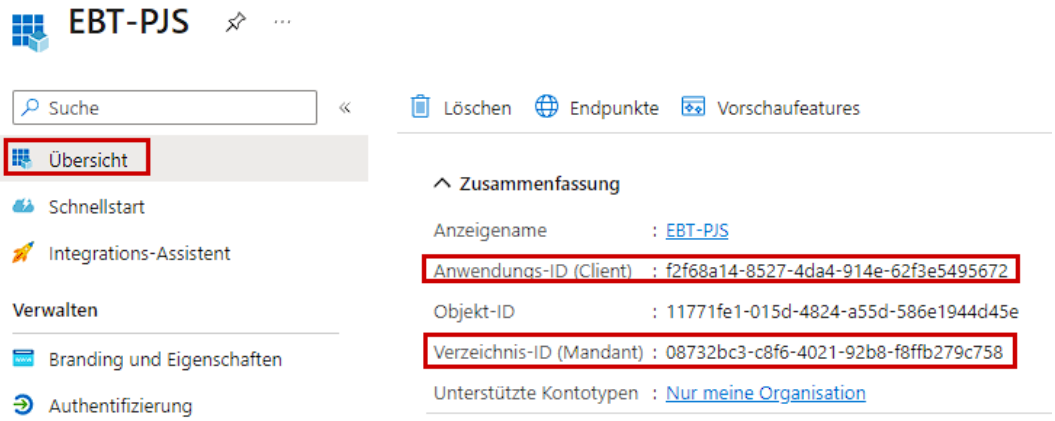

Abbildung 4: ClientID und TenantID in Azure

Damit ist die Anbindung von Microsoft Graph abgeschlossen.

5 Kafka-Streaming Server

Als letzter Schritt wird der Kafka Streaming Server installiert und als Service aufgesetzt. Abschließend wird gezeigt, wie mittels eines Linux-Cronjobs stündlich Daten von OpenWeatherMap angefordert werden können.

5.1 Installation und Service aufsetzen

Folgender Code wird ausgeführt, um den Kafka-Streaming Server zu installieren:

```
apt install openjdk-8-jre-headless
cd /usr/local/
wget https://archive.apache.org/dist/kafka/3.3.1/kafka_2.13-3.3.1.tgz
tar -xzf kafka_2.13-3.3.1.tgz
mv kafka_2.13-3.3.1 kafka
cd /etc/systemd/system
nano zookeeper.service
# Text siehe Kapitel
nano kafka.service
# Text siehe unten
```

In der erstellten Datei zookeeper.service wird folgender Text eingefügt:

```
[Unit]
Description=Bootstrapped Zookeeper
After=syslog.target network.target
[Service]
Type=simple
User=root
Group=root
Restart=on-failure
RestartSec=5s
ExecStart=/usr/local/kafka/bin/zookeeper-server-start.sh
/usr/local/kafka/config/zookeeper.properties
ExecStop=/usr/local/kafka/bin/zookeeper-server-stop.sh
[Install]
WantedBy=multi-user.target
```

In der erstellten Datei kafka.service wird folgender Text eingefügt:

```
[Unit]
Description=Apache Kafka
Requires=zookeeper.service
After=zookeeper.service
[Service]
Type=simple
User=root
Group=root
Restart=on-failure
RestartSec=5s
```

```
ExecStart=/usr/local/kafka/bin/kafka-server-start.sh
/usr/local/kafka/config/server.properties
ExecStop=/usr/local/kafka/bin/kafka-server-stop.sh
[Install]
WantedBy=multi-user.target
```

Abschließend wird der Service gestartet:

```
chmod +x zookeeper.service
chmod +x kafka.service
systemctl daemon-reload
systemctl enable zookeeper.service
systemctl start zookeeper
systemctl enable kafka.service
systemctl start kafka
```

5.2 Cronjob initalisieren

Nach dem Aufsetzen des Kafka-Streaming Servers wird ein Cronjob aufgesetzt, um in fest definierten Zeitintervallen den Kafka-Producer und Kafka-Consumer auszuführen. Die Crontab-Konfigurationsdatei kann über den Linux-Befehl **crontab -e** aufgerufen werden. Dort müssen die folgenden beiden Zeilen ergänzt werden:

```
0 * * * * /usr/bin/python
/var/www/PJS/streaming_data_platform/producer_weather.py

10 * * * * /usr/bin/python
/var/www/PJS/streaming_data_platform/consumer_weather.py
```

6 Abschluss

Durch Eingabe der Server URL im Browser wird nun initial die Registrierungsseite aufgerufen. Nachdem ein Nutzeraccount angelegt worden ist, kann die gesamte Funktionalität des Portals genutzt werden.