Abstract

Im Rahmen dieser Arbeit wird die Arbeit von Herren Tomak "Performance-Testing-Infrastruktur" (2020) erweitert.

//TODO
schreib über die Arbeit und über Locust.

Die Erweiterung umfasst die Implementierung eines Web-Services, das Benutzern/Benutzerinnen die Möglichkeit anbietet, Load-Testing ihres gewählten zu testenden Systems nach dem Festlegen der Testspezifikationen und dem Hochladen der durchzuführenden Test-Datei durchzuführen, wobei Locust bevorzugt wird.
Außerdem muss das zu implementierende Web-Service einfach zu installieren und zu benutzen sein. Um dieses Ziel zu realisieren, wird der Kern des Web-Services als eine Openfaas-function implementiert.

Openfaas ist ein Opensource-Projekt, das von *Alex Ellis* mithilfe mehrerer Mitwirkenden entwickelt wurde. Das Ziel des Projekts ist es, Serverless-Functions in einer der derzeit unterstützten Programmiersprachen zu schreiben und unabhängig vom Betriebssystem benutzen zu können. Die derzeit unterstützten Programmiersprachen sind unter anderem *Python, Java, C Sharp, Go* und vieles mehr. Eine Liste der unterstützten Programmiersprachen ist unter

https://github.com/openfaas/templates

zu finden. Beim Erstellen einer Openfaas-function wird ein Docker-Container mit allen Abhängigkeiten, die die Funktion benötigt, automatisch erstellt. Diese Abhängigkeiten werden von Entwickler/Entwicklerinnen in einer Requirements-Datei aufgelistet. Konfigurationen der Funktion werden in einer yaml-Datei organisiert. Erstellen und Einsetzen einer Funktion sind Aufgaben des Programms *faas-cli* mithilfe von laufendem Docker-Engine.

Weitere Information über Openfaas sind unter

https://github.com/openfaas/faas

https://docs.openfaas.com/

https://github.com/openfaas/faas-cli

zu finden.

Das Web-Service soll auch Folgendes anbieten:

- Mehrere Tests gleichzeitig durchführen.
- Laufende Tests stoppen.
- Laufende und nicht laufende Tests löschen.
- Einen Überblick über die Lage aller Tests anbieten. Der Überblick enthält Informationen über den Test sowie Status (laufend, noch nicht laufend und fertig ausgeführt), Laufzeit, Testspezifikationen, Sourcecode des Testes und aktuelle Statistiken der Tests.
- Visualisierungen anzeigen sowie Liniendiagramme der Antwortzeit und Lineare Regression der Antwortzeiten.
- Testergebnisse zusammenstellen und zum Herunterladen anbieten.
- Direkte oder indirekte Kommunikation zwischen dem Browser und Openfaas anbieten.
    - Direkte Kommunikation: der Browser schickt Anfragen direkt an Openfaas und bekommt direkte Antworten zurück.
    - Indirekte Kommunikation: der Browser schickt Anfragen an den Server, der diese an Openfaas weiterleitet und Antworten von Openfaas an den Browser zurück schickt.

  Diese beiden Varianten der Kommunikation liefern eine größere Flexibilität des Web-Services sowie einen Gewinn an Reaktionszeit des Web-services im Falle des Einsetzens der direkten Kommunikation.

  Anwendungsfälle der beiden Varianten der Kommunikation:
    - Direkte Kommunikation: Openfaas läuft auf einem Host, der externe Anfragen annehmen kann.
    - Indirekte Kommunikation: Openfaas läuft auf einem Host, der keine externen Anfragen annehmen darf. Der Server läuft auf einem Host im selben Netzwerk, der aber externe Anfragen annehmen kann.

Das Implementieren des Web-Services erfolgt mithilfe folgender Programmiersprachen bzw. Frameworks:

- **Frontend**: Javascript, Html und Css.
- **Backend**: Python-flask
- **Openfaas-function**: Python-flask

Struktur des Web-Services:

- **ptas**
  - **handler.py**
    Definition der Openfaas-function.
  - **requirements.txt**
    Python-Bibliotheken, die die Openfaas-function benötigt.
- **template**
  Vorlage der Openfaas-function.
- **server**
  - **static**
    - **css**
      Design der Webseite.
    - **scripts**
      Benötigte Javascript-Dateien.
  - **templates**
    Html-Dateien, die gerendert werden.
  - **server.py**
    Main-Datei.
- **ptas.yml**
  Openfaas-function-Konfigurationsdatei.
- **requirements.txt**
  Python-Bibliotheken, die der Server benötigt.

Ausführen des Webservices:

- Einsetzen der Openfaas-function.
  Nach dem Konfigurieren der Yaml-Datei erfolgt das Einsetzen der Funktion mit dem folgenden Befehl.

  ```
  faas-cli up -f ptas.yaml
  ```

  wobei Docker-Engine auf dem Rechner laufen muss.

- Installieren der benötigten Python-Bibliotheken.
  Die Installation der benötigten Python-Bibliotheken erfolgt ganz einfach mit dem folgenden Befehl.

  ```
  pip install -r requirements.txt
  ```

- Ausführen der Main-Datei.
  Das Ausführen der Main-Datei erfolgt mit dem folgenden Befehl:

  ```
  python server.py -s <host> -p <port> -u <openfaas-url> -f
  <openfaas-function-name> -d <true || false>
  ```

  Wobei diese Argumente notwendig sind.

  Im Folgenden sind alle Argumente der Main-Datei aufgelistet:

  ```
  optional arguments:
    -h, --help          help
    -v, --version       version

  required arguments:
    -s , --host         server host
    -p , --port         server port
    -u , --url          openfaas url
    -f , --function     function name
    -d , --direct       can the browser connect to openfaas
                        directly? <true || false>
  ```