Aufgabe 5: Implementierung des Chatservers

Bisher wurde Ihnen der Chatserver für alle Aufgaben als Node-RED Flow bereitgestellt. In dieser Aufgabe sollen alle Funktionen dieses Servers als *Node.js* Anwendung auf Basis von *Express* (siehe Tafelübung) implementiert werden. Der Webserver *nginx*[1] kommt dabei als *Reverse Proxy* (siehe Tafelübung) zum Einsatz. Zusätzlich beinhaltet der Chatserver eine MongoDB Instanz, um Daten persistent halten zu können. Am Ende der Aufgabe sollen diese Komponenten in Docker-Containern auf einer *virtuellen Maschine* (VM) in der vom i4 betriebenen *OpenStack*[2] (siehe Tafelübung) ausgeführt werden.

5.1 Lokale Ausführung des Servers und Vorbereitung

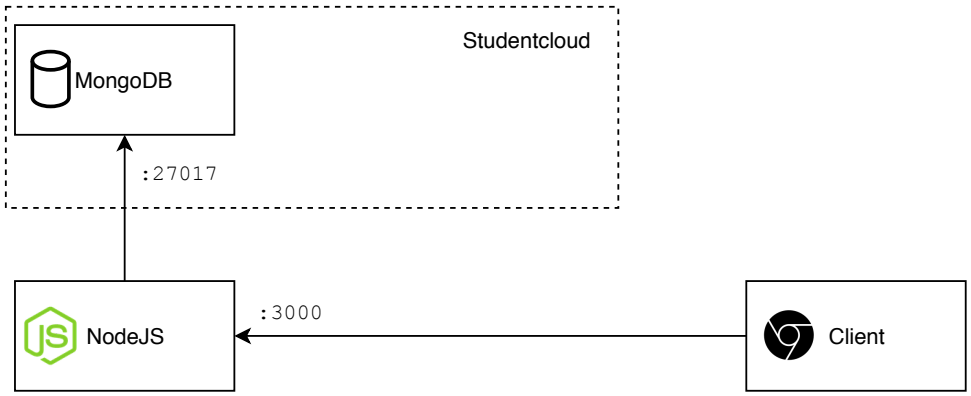

Abbildung 1: Deployment des Chatservers während der Entwicklung

In Ihrem Repository finden Sie im Ordner `05_server/chat_server` mehrere Dateien, auf deren Basis Sie den Chatserver implementieren sollen. Wie in Abbildung 1 dargestellt, soll während der Implementierung nur die MongoDB-Instanz in der VM laufen. Wenn Sie die Aufgabe auf Ihren eigenen Rechner bearbeiten, müssen sie die OpenStack VM nicht für die MongoDB verwenden, starten Sie dazu den entsprechenden Docker Container lokal.

- Installieren Sie alle benötigten Abhängigkeiten mit dem Befehl `npm install` und starten Sie anschließend den Server mit `npm start`.
- Verifizieren Sie durch Aufruf von `http://localhost:3000`, dass der Server die Datei im Ordner `public/` korrekt ausliefert.
- Entfernen Sie die ausgelieferte Datei `public/index.html` aus dem Repository und legen Sie stattdessen *symbolische Links* (abgekürzt *Symlinks*, siehe `man ln` unter Linux bzw `mklink` unter Window) an, sodass der Server den von Ihnen in den letzten Aufgaben implementierten Client ausliefert. Die auszuliefernden Dateien sollen *nicht* kopiert werden! Fügen Sie die Symlinks auch ihrem git Repository hinzu!

Zusätzliche Hinweise:

- Verwenden Sie keine *Hard Links* (siehe `man ln`)!
- Verwenden Sie *relative* Pfade für die symbolischen Links!

5.2 Einrichtung des Back-Ends in der OpenCloud

Die quelloffene Software *OpenStack* stellt eine Architektur für das sogenannte Cloud-Computing zur Verfügung. Damit lassen sich unter anderem *virtuelle Maschinen* schnell zugänglich machen. Das i4 betreibt eine eigene OpenStack Instanz.

5.2.1 Start einer virtuellen Maschine und Zugriff

Wie bereits beschrieben soll zunächst die MongoDB Instanz und später der Chatserver auf einer virtuellen Maschine (VM) in der OpenCLoud ausgeführt werden (siehe Aufgabe 5.4).

- (Optional) erstellen Sie einen SSH-Schlüssel. Benutzen Sie dafür `ssh-keygen`.

[1] `https://nginx.org/`
[2] `https://i4cloud1.cs.fau.de/`

- Starten Sie eine VM, indem Sie unter Compute → Instanzen auf + Neue Instanz klicken. Wählen Sie dabei als Boot-Quelle "von Abbild starten" und wählen Sie das Abbild "WebSys_Ubuntu" aus. Als Variante (Flavor), wählen Sie *i4.docker* aus. Fügen Sie *vor* Start der VM unter dem Reiter "Key Pair" einen öffentlichen SSH-Schlüssel hinzu, der in Ihrer VM hinterlegt werden soll.
- Starten Sie die Instanz und weisen Sie ihr anschließend eine öffentliche IP-Adresse (eine sogenannte *Floating IP*) zu, um sie von außen erreichbar zu machen. Loggen Sie sich anschließend mit dem Kommando `ssh ubuntu@<floating-ip>` ein. Die Floating IP und weitere Eigenschaften Ihrer VM können Sie jederzeit der Tabelle "Instanzen" entnehmen.
- Erlauben Sie schließlich auch SSH-Zugriff für Ihr Teammitglied. Erstellen Sie dazu die Datei `~/.ssh/authorized_keys` auf der VM und fügen Sie eine *Zeile hinzu*, welche den entsprechenden öffentlichen Schlüssel der folgende Form enthält:

  ```
  ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDDQx... user@host
  ```

  Den öffentlichen Schlüssel Ihres Teammitglieds finden Sie üblicherweise in der Datei `~/.ssh/id_rsa.pub` im entsprechenden Home-Verzeichnis.
- VMs unterliegen gewissen Zugriffsbeschränkungen auf der Basis von Ports. Diese werden in *Sicherheitsgruppen/Security Groups* definiert. Standardmäßig ist **nicht mal** Port 22 für das SSH-Protokoll geöffnet. Erlauben sie zu dem SSH-Potokoll zusätzlich die Ports 80 (HTTP) und 27017 (MongoDB), indem Sie entsprechende Regeln zu einer neuen "neuen" Sicherheitsgruppe hinzufügen. Die Einstellungen dazu finden Sie unter Compute → Zugriff & Sicherheit. Verwenden Sie als Remote den IP Bereich (CIDR) das Netz `0.0.0.0/0`.

Zusätzliche Hinweise:

- Wenn Sie eine Instanz *terminieren / beenden* geht sämtlicher Zustand verloren. Wählen Sie stattdessen *abschalten* bzw. *shut off*. Wenn Sie Ihre Maschine doch terminieren möchten, sollten sie vorher eine(n) *Snapshot / Schattenkopie* erstellen, um den Zustand zu sichern.
- Sie können Dateien mit `scp` oder `rsync` auf Ihre VM kopieren.

5.2.2 Start der MongoDB Instanz

Klonen Sie Ihr git Repository in der VM, um Ihre aktuelle Implementierung auf dem Server bereitzustellen. Am einfachsten ist hierbei ein `git clone` über HTTPS, da hier kein SSH-Key benötigt wird. Führen Sie im Ordner `05_server` den Befehl `sudo docker-compose up mongodb` aus, um die MongoDB Instanz zu starten. Dies ist erfolgreich, wenn Sie eine Ausgabe ähnlich zu `waiting for connections on port 27017` sehen. Diese Nachricht ist nicht notwendigerweise die letzte Nachricht von MongoDB.

5.3 Implementierung des Servers mit Node.js

In dieser Teilaufgabe sollen Sie die aus den vorherigen Übungen bekannten Serverfunktionalitäten selbst mit Node.js implementieren.

5.3.1 Persistente Datenhaltung mit MongoDB

Der Chatserver muss den gespeicherten Chatverlauf persistent halten können, d.h. die Daten dürfen nach einem Neustart des Servers nicht verloren gehen. Verwenden Sie dazu die in der Tafelübung vorgestellte quelloffene NoSQL-Datenbank *MongoDB*[3].

- Implementieren Sie in der Datei `models/message.js` ein Node.js Modul, welches das Objekt `Message` exportiert. Dieses Objekt soll ein Mongoose[4] Model (siehe Tafelübung) sein.
- Die URL, unter der die MongoDB erreichbar ist muss von außen konfigurierbar sein, werten Sie daher den Inhalt der Umgebungsvariable (siehe Tafelübung) `DB_URL` aus, um eine Verbindung zur MongoDB herzustellen.

[3] `https://www.mongodb.com/`
[4] `https://mongoosejs.com/`

5.3.2 Implementierung eines REST Interfaces

Es soll ein REST Interface erstellt werden, das JSON-basiert ist. Dementsprechend nimmt das Interface ausschließlich JSON-Objekte an und gibt ausschließlich JSON-Objekte zurück. Das im Repository hinterlegte "Gerüst" des Servers enthält die Datei `routes/chat.js`; legen Sie hier alle Routen (siehe Tafelübung) an. Dabei soll `routes/chat.js` ein Node.js Modul (siehe Tafelübung) implementieren!

- Erweitern Sie den Server um eine Express Route für HTTP `POST` Requests mit der Location `/chat`. Hier erwartet der Server im Body des Requests ein JSON-Objekt mit der Nachricht. Der Inhalt des Objekts soll über das Message Objekt (siehe Aufgabe 5.3.1) in der MongoDB hinterlegt werden.
- Erweitern Sie den Server um eine Express Route für HTTP `GET` Requests mit der Location `/savedMessages`. Die HTTP Reply soll alle in der MongoDB hinterlegten Nachrichten als JSON-Objekt zurückgeben. Das Format des JSON-Objekts soll dabei kompatibel zu den vorherigen Aufgaben sein.

Zusätzliche Hinweise:

- `routes/chat.js` soll ein Node.js Modul (siehe Tafelübung) implementieren!
- Da das von Ihnen zu implementierende REST Interface nur JSON spricht kann ein Browser nicht immer verwendet werden. Nutzen Sie deshalb das Kommandozeilenprogramm `curl` bzw. die Entwicklertools von Firefox oder Chrome.

5.3.3 WebSockets

Fügen Sie ihrer Implementierung Support für WebSockets hinzu, benutzen Sie dazu die in der Tafelübung vorgestellte Bibliothek `express-ws`[5].

- Erweitern Sie die Datei `routes/chat.js` um eine Route für WebSockets unter der Location `/ws`.
- Sorgen Sie dafür, dass eingehende Nachrichten in der MongoDB hinterlegt werden.
- Implementieren Sie das *Broadcasting* von Nachrichten: Eingehende Nachrichten sollen an alle verbundenen Clients weitergleitet werden.
- Sie müssen Ihren Client-Code so ändern, dass er eine Verbindung zum Server für die WebSocket-Verbindung herstellt, anstatt sich mit localhost zu verbinden.

5.3.4 Lokaler Test des Servers

Testen Sie Ihre Serverimplementierung ausgiebig in der lokalen Konfiguration (siehe Abbildung 1), bevor Sie mit Aufgabe 5.4 weitermachen. Starten Sie dazu den Server mit der in 5.3.1 erwähnten Umgebungsvariable: `DB_URL=mongodb://[floating-ip]/chatDB npm start`.

5.4 Deployment des Servers in der Studentcloud

In der letzten Teilaufgabe sollen alle Komponenten, wie in Abbildung 2 dargestellt, in der OpenCloud ausgeführt werden.

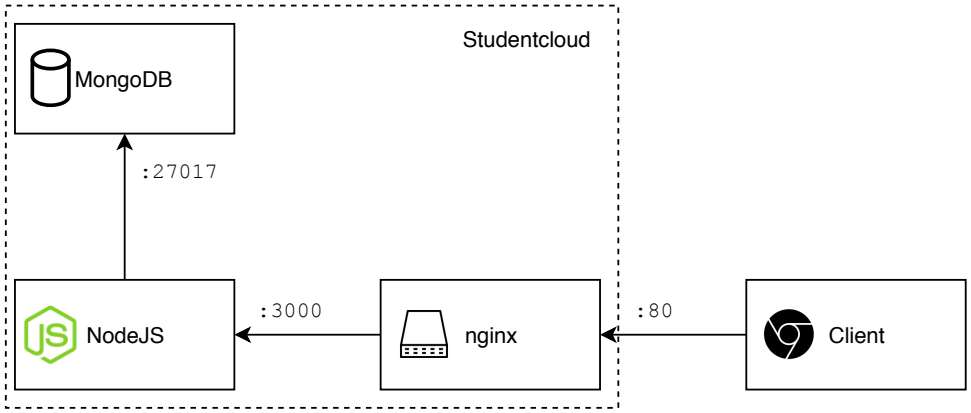

Abbildung 2: Endgültiges Deployment des Chatservers mit nginx als Reverse Proxy

[5] `https://www.npmjs.com/package/express-ws`

- Stellen Sie zuerst Ihre Änderungen auf der VM über Ihr git Repository zur Verfügung.
- Schließen Sie den in Aufgabe 5.2 geöffneten Port für MongoDB, sodass Ihre VM von außen nur noch HTTP und SSH zulässt.
- Es werden die folgenden drei Dockercontainer zur Verfügung gestellt:
  - `mongodb`: Startet eine MongoDB Instanz, welche auf Port 27017 lauscht (bereits in Aufgabe 5.2.2 verwendet)
  - `nginx`: Startet den Webserver nginx, konfiguriert als Reverse Proxy (leitet HTTP-Anfragen von Port 80 an Chatserver Port 3000 weiter)
  - `node`: Startet den von Ihnen implementierten Server (führt `npm start` in `chat_server` aus)

  Diese Container sind in der Datei `docker-compose.yml` definiert. Erstellen Sie alle Container mit `sudo docker-compose up --no-start` und starten Sie sie anschließend mit `sudo docker-compose start`.
- Testen Sie ihr Deployment!

Zusätzliche Hinweise:

- Die Docker-Container können sich untereinander durch Hostnames erreichen. Die Hostnames entsprechen dabei dem Namen des Containers, der in der `docker-compose.yml` definiert ist.
- Mit `sudo docker-compose build` können Sie die Container neu bauen, nachdem sie an den Quelldateien Änderungen gemacht haben.

Letzter Commit bis zum 05. Februar.
Termine für die Vorstellungen per Terminklick ab 05. Februar.

Präsentation der fertigen Lösung spätestens am Tag der Abgabefrist in der Rechnerübung!