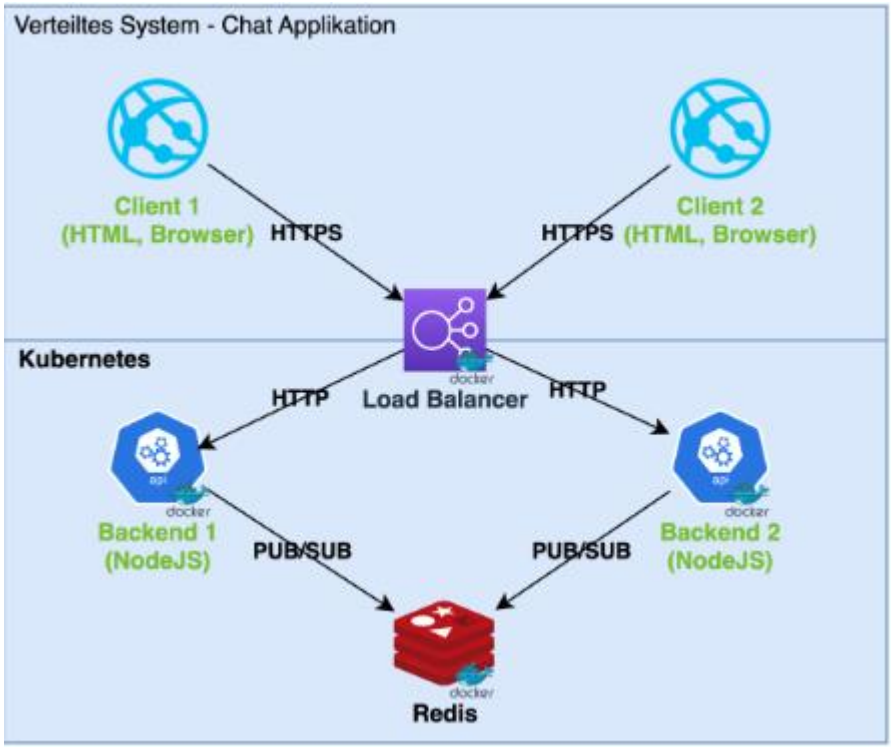
Autor: Gabriel Bühler

Transferarbeit Chatapplikation



Inhalt

Einleitung

Diese Arbeit dient dazu aufzuzeigen, wie Systeme aufgeteilt werden können, um deren Flexibilität Skalierbarkeit und Ausfallsicherheit zu verbessern. Dafür werden Anfragen über einen Load-Balancer an mehere Backends verteilt, die dann schlussendlich über einen Messagebroker kommunizieren können.

Client

Bei diesem Projekt handelt es sich beim Client um eine HTML-Seite, die mit Hilfe von Javascript-Code als Webapplikation im Vordergrund für den Nutzer agiert. Der User hat dadurch die Möglichkeit, Nachrichten zu versenden und zu empfangen, die werden dann über einen Load-Balancer an das

Backend verteilt. Zudem werden alle User, die derzeit auf den Server zugreifen mit Namen angezeigt Dies geschieht durch ein Script, dass mit einem guidGenerator einen zufälligen Identifier erstellt und dann über die «Random User API» einen Benzutzernamen erstellt. Diese Nachrichten sendet es dann über den «Message»-Event an den Websocket Server. Ausserdem kann es erhaltene Nachrichten auf vom Websocket erhalten und anzeigen, indem es auch ein Objekt mit dem Typen «Message» erhält und Anhand der benötigten Informationen (User ID, UserName, Message, Time) die Nachricht als HTML-Code wieder ausgeben kann. Es zeigt zudem auch an, wenn es von dem WebsocketServer die Benachrichtigung erhält, dass eine Session geschlossen wird. Die Verbindung zu dem Backend Websocket-Server aufbaut über Der Loadbalancer ist hier ein Lösungsansatz, um für eine Ausfallsicherheit zu sorgen, da dadurch erreicht werden kann, dass immer mindestens ein System erreichbar ist, auch wenn eines der Komponenten im Backend nicht erreichbar sein sollten (z.B. Aufgrund von Updates oder Systemproblemen.).

Backend

Der WebsocketServer Express wird hier aufgebaut, der die erhaltenen Nachrichten weiter an den Message Broker «Redis» sendet. Dieser aktualisiert sich automatisch neu nach jeder neuen Anfrage eines clients. Zudem initialisiert es eine REST-API, die als Schnittstelle zwischen dem Redis Message Broker und der Frontend-Applikation agiert. Hier können mehrere Instanzen dieses Servers «hinter» den Load Balancer gesetzt werden, der dann die Anfragen an die verschiedenen Server verteilen kann, dass im Falle eines Ausfalles der andere Server übernehmen kann.

Message Broker

Der Messagebroker ist für die Asynchrone Kommunikation zwischen den einzelnen Websocket Server und der Frontend-Applikationen zuständig. Dabei können die einzelnen Clients, die auf den Dienst zugreifen jeweils als Publisher und Subscriber agieren. Die «Publisher» sendet dabei die Nachricht (in unserem Falle eine Chatnachricht) an den Message Broker, der dann die Nachricht an alle Clients verteilt, die über das gleiche Topic identifiziert wurden. Die Nachrichten erhält es auch vom Websocket Server, Inform eines Events, in unserem Falle den «Message»-Event. Wenn es einen «Close»-Event erhält, wird die Websocket-Verbindung abgebrochen und die Applikation wird geschlossen. Die Ausfallsicherheit von Redis kann mit dem «Scaling-Mode» verbessert werden, dadurch ist es entweder möglich, mehrere Server (Nodes) zu verwenden, die die Anfragen auch über Load-Balancers annehmen (Horizontal Scaling), oder der Server kann je nach Bedarf automatisch mehr Arbeitsspeicher und Prozessleistung des Hostsystems zugesprochen bekommen (Vertical Scaling).

Ausfallsicherheit

Das gesamte System kann zusätzlich noch ausfallsicherer gebaut werden, in dem die einzelnen Systeme mit einer Container-Orchestrierung wie Kubernetes verteilt werden. Dadurch ist es sehr viel einfacher, die einzelnen Systeme zu skalieren, in die zusätzlichen Systeme als sogenannte «Nodes» dazu genommen werden, die dann in einem «Cluster» gruppiert werden. Auf denen dann die Applikation als Container aktiv ist. Dort kann man die Skalierung auch mit «Autoscaling» automatisieren, bei dem automatisch nach Bedarf zusätzliche Nodes dazugeschaltet werden können. Dadurch ist eine gute Ausfallsicherheit garantiert, da die anderen Nodes weiterlaufen, falls einer davon aufgrund eines Problems ausfallen sollte.

Testprotokoll

Testname	Vorgang	Erwünschtes Resultat	Resultat erreicht?
Versenden einer Nachricht	Eine Nachricht wird in das Textfeld eingegeben.	Die eingegebene Nachricht wird im Chatfenster angezeigt, mit dem Benutzernamen des Users und einem Zeitstempel	Ja
Erhalten einer Nachricht	Die anderen User erhalten die gesendete Nachricht ebenfalls	Die anderen User erhalten die gesendete Nachricht mit dem Usernamen und dem Zeitstempel der Person	Ja
Ändern des Benutzernamens	Der Benutzername kann geändert werden	Der neue Benutzername wird angezeigt und für alle User aktualisiesrt	Ja
Deployment des Programmes auf Docker	Das Programm kann als Docker-Image erstellt und deployed werden, alles funktioniert wie gewüsnscht	Die Applikation läuft in einem Docker-Container und kann genutzt werden	Ja
Verteilung der Container über Kubernetes	Die Container können über Kubernetes genutzt werden	Die Applikation ist aktiv über Kubernetes auf der Webseite buehler.teko.hackerman.ch	Ja