Übung 2

Professionelle Web-Anwendungen umsetzen und betreiben

Sommer Semester 2023
Berliner Hochschule für Technik (BHT)

B.Sc. Medieninformatik

Kelvin Zihang Kuang (925597)

Inhaltsverzeichnis

Einleitung

Im Rahmen des Moduls "Professionelle Webanwendungen umsetzen und betreiben" wird eine CI-Pipeline für die Webanwendung entwickelt, die im Modul "Web Engineering 2" erstellt wurde. In der vorherigen Übung wurden bereits End-to-End-Tests mit dem Framework Playwright geschrieben. Die Testumgebung wurde parallel zur Webanwendung in ein öffentlich einsehbares Repository auf Github.com hochgeladen.

Die Strategie für den Entwurf dieser CI-Pipeline wurde ausschließlich von Kelvin Zihang Kuang (925597) entwickelt.

Ziel der CI-Pipeline

Eine CI-Pipeline dient der kontinuierlichen Integration von neuem Code in eine Software. Um die Qualität der Software zu gewährleisten, müssen Maßnahmen ergriffen werden, um fehlerhaften Code zu vermeiden.

Die Tests aus der vorherigen Übung eignen sich daher besonders gut. Allerdings können Testläufe, insbesondere bei End-to-End-Tests, langsam sein und den Arbeitsablauf in einer professionellen Umgebung stark beeinträchtigen.

Daher ist es sinnvoll, solche Qualitätskontrollen zu automatisieren. Eine gut durchdachte CI-Pipeline ist daher ein wesentlicher Bestandteil einer Software.

Github.com bietet auf seiner Plattform Entwicklertools an, die den Entwurf solcher CI-Pipelines erheblich vereinfachen. In Github wird ein ".github"-Verzeichnis im gewünschten Repository erstellt, das von der Plattform erkannt wird, um auf bestimmte Ereignisse zu reagieren.

Vorbereitung vor dem Entwurf.

Bei der Entwicklung der Webanwendung wurde zunächst ein REST-Server entworfen. Allerdings wurde während der Entwicklung vom Professor ein zuverlässiger REST-Server bereitgestellt. Dieser wurde weiterhin für die Entwicklung des Frontends und der Testumgebung verwendet.

Da der bereitgestellte REST-Server eine ausführbare Datei ist, gestaltete sich die Integration in die CI-Pipeline als schwierig. Es wurden umfangreiche Recherchen durchgeführt, die jedoch den Rahmen dieser Übung sprengen.

Bei der Verwendung des eigenen Backend-Servers musste zunächst manuell sichergestellt werden, dass er sich nicht in einem fehlerhaften Zustand befindet. Die End-to-End-Tests waren dabei sehr hilfreich, da sie aufzeigten, dass bei den Studienbewerbungen einige Daten fehlten.

Außerdem scheitern zwei Tests ständig, wenn der eigens entwickelte Backend verwendet wird, und das Beheben dieser Fehler würde den Zeitrahmen dieser Übung sprengen. Daher wurden diese beiden Tests vorerst auskommentiert. Falls nach dem Entwurf und der Überprüfung der CI-Pipeline Zeit zur Verfügung steht, werden die Fehler behoben. In einer professionellen Umgebung sollten solche Fehler mit höchster Priorität schnell behoben werden, um unvorhergesehene Probleme in der Produktionsumgebung zu vermeiden.

Die nächste Vorbereitung für den Entwurf der CI-Pipeline besteht darin, sich klarzumachen, was im Container geschieht. Die entwickelte Webanwendung basiert auf dem MERN-Stack, der jedoch ausschließlich lokal entwickelt wurde. Daher gibt es keine festen URLs, mit denen der Stack verbunden werden kann.

Dies ist insbesondere für die MongoDB-Datenbank relevant. Im Backend wurde lediglich die Mongoose-Schnittstelle zwischen Node.js und MongoDB verwendet, und es wurde keine eigenständige Instanz der Datenbank erstellt. Die Datenbank war eine lokal installierte MongoDB-Anwendung, die nicht in das Repository aufgenommen wurde.

Daher ist es ein wichtiger Schritt im Workflow, die Installation und den Start einer MongoDB-Instanz in die CI-Pipeline einzubeziehen. Hierfür mussten die entsprechenden Befehle recherchiert und getestet werden, da es üblich ist, in einem Container die aktuelle Ubuntu-Version zu verwenden.

Es ist außerdem anzumerken, dass dieser Schritt in der Realität kein Teil der CI-Pipeline sein kann, da die Datenbank selbst nicht Teil der Software ist. Stattdessen sollte die Anbindung an die Datenbank innerhalb der CI-Pipeline-Umgebung ein Bestandteil der Tests sein, die überprüft werden sollten.

Beschreibung der Schritte, um die Webanwendung zu starten

Die CI-Pipeline bietet eine automatisierte Möglichkeit, die Webanwendung zu starten. Daher ist es zunächst wichtig, die einzelnen Schritte zum Starten der Webanwendung zu beschreiben.

Der allererste Schritt besteht darin, die MongoDB-Instanz zu starten. Im Fall der lokalen Entwicklung wurde dies bereits durchgeführt. In einem Ubuntu-Container müssen die entsprechenden Ubuntu-Befehle ausgeführt werden, um die MongoDB-Instanz im Container aus dem Netzwerk zu installieren.

Als nächstes müssen die erforderlichen Abhängigkeiten für das Backend installiert werden, da Systembibliotheken nie in ein Git-Repository eingepusht werden sollten. Nach der Installation kann das Backend mit dem Befehl "npm start" gestartet werden und im Hintergrund ausgeführt werden.

Analog dazu müssen die gleichen Schritte für das Frontend durchgeführt werden. Da das Frontend mehr Systemabhängigkeiten hat, dauert die Installation entsprechend länger. Gleiches gilt für das Starten des Frontend-Servers.

Auf meinem persönlichen Entwicklungsrechner dauert das Starten der Server nur Sekunden, aber ich bin mir bewusst, dass es auf anderen Systemen deutlich länger dauern kann.

Dies ist der Grund, warum im Workflow eine bestimmte bedingte Reihenfolge der Abläufe erforderlich ist.

Mit diesen Schritten läuft die Webanwendung unter localhost:3000 und funktioniert. Nun muss die Testumgebung gestartet werden. Dafür müssen auch hier die erforderlichen Abhängigkeiten installiert werden. Sobald diese installiert sind, kann mit dem Befehl "npm test" die Testsequenz gestartet werden.

Die Strategie der CI-Pipeline orientiert sich an diesem Schema. All dies befindet sich innerhalb einer Stage, die im folgenden Workflow als "test" bezeichnet wird.

Der Workflow

Beim Einrichten der CI-Pipeline müssen die oben genannten Schritte in Code übersetzt werden, damit der Container weiß, welche Schritte erforderlich sind, um die Webanwendung zu starten.

Zunächst müssen jedoch einige Rahmenbedingungen definiert werden. Github Actions lauscht auf verschiedene Events im ".github"-Verzeichnis des Repositories. Eines der häufigsten Events, auf die gehört wird, ist das "Push Event". Darüber hinaus muss festgelegt werden, auf welchem Branch Github reagieren soll. In meinem Fall gibt es nur den Hauptbranch namens "master", daher lauschen alle Events darauf.

Es macht Sinn, die Pipeline auszuführen, sobald neuer Code in die Haupt-Codebase gepusht wird, da ein fehlerhafter Merge an dieser Stelle schwerwiegende Auswirkungen haben könnte. Allerdings fordert die Aufgabenstellung ausdrücklich, dass die Pipeline bei jedem Commit ausgeführt wird.

Nachdem dies spezifiziert wurde, können die einzelnen Schritte in der YAML-Datei beschrieben werden. Zunächst muss dem Workflow angegeben werden, dass er den Code aus dem Repository auschecken soll, damit er weiß, in welcher Umgebung er arbeiten soll.

Als nächstes muss die MongoDB-Instanz mit einem Ubuntu-Bash-Skript installiert und gestartet werden. Um sicherzustellen, dass die MongoDB-Instanz ordnungsgemäß gestartet wurde, wurde ein kleines Skript erstellt, das überprüft, ob sie läuft.

Analog dazu müssen die Abhängigkeiten für die Backend- und Frontend-Server mit dem Befehl "npm ci" installiert und mit dem Befehl "npm start" gestartet werden. Auch hier wurde ein kleines Skript verwendet, um sicherzustellen, dass die entsprechenden Server ausgeführt werden.

Aufgrund meiner Unerfahrenheit mit Bash-Skripten musste ich mich beim Schreiben der Skripte auf das Tool ChatGPT verlassen. Das Erlernen von Bash-Skripten ist zwar äußerst nützlich, stand jedoch nicht im Mittelpunkt dieser Übung.

ChatGPT bietet nun die Möglichkeit, vergangene Konversationen zu speichern. Die entsprechenden Konversationen wurden als Snapshot heruntergeladen und im Quellverzeichnis vermerkt, um Fehler entsprechend nachverfolgen zu können.

Probleme und Fehler der CI-Pipeline

Beim Implementieren des Workflows traten einige Fehler auf. Die erste große Hürde bestand darin, dass die Ubuntu-Umgebung aufgrund fehlender Berechtigungen das Starten

des Backend-Servers verhinderte. Eine kurze Recherche [1] ergab, dass nicht privilegierte Benutzer keine "Listening Sockets" unter Port 1024 öffnen können.

Eine mögliche Lösung für diesen Fehler wäre die Verwendung einer höheren Portnummer für die entsprechenden Ports. Beim ersten Versuch führte dies jedoch dazu, dass die gesamte Anwendung nicht mehr ordnungsgemäß startete. Das Beheben dieses Problems brachte keine Garantie dafür, dass dieser Fehler nicht erneut auftritt. Daher entschied ich mich dafür, das Starten der Server mit dem Schlüsselwort "sudo" durchzuführen, um die Server im Namen eines Administrators zu starten.

Nachdem diese Fehler behoben waren, setzte die CI-Pipeline ihren Ablauf fort bis zum Schritt, in dem die Tests automatisch durchgeführt werden sollten. Vor der Implementierung des Workflows wurde sichergestellt, dass alle End-to-End-Tests lokal erfolgreich durchgeführt wurden. In der CI-Pipeline schlugen jedoch alle Tests fehl.

Eine Maßnahme bestand darin, die Bash-Skripte anzupassen, um sicherzustellen, dass die Server und die Datenbank ordnungsgemäß liefen. Hierbei wurden "curl"-Befehle verwendet, um auf eine Antwort der jeweiligen Server zu warten.

Nachdem sichergestellt wurde, dass die Pipeline nicht an den Servern scheiterte, wurden weitere Versuche unternommen, um die Pipeline erfolgreich ablaufen zu lassen.

Nach Rücksprache mit dem Professor erhielt ich den Hinweis, auf bestimmte Elemente zu warten, bevor die Tests fortgesetzt werden. Ich verglich diesen Ansatz mit den Sourcecodes von Playwright-Tests, die ich während meiner aktuellen Tätigkeit bei "Sprylab Technologies" entwickelt habe, und stellte fest, dass dies nicht notwendig ist.

Der End-to-End-Test-Entwickler bei Sprylab wies darauf hin, dass es bei Playwright nicht unbedingt erforderlich ist, auf jedes Element zu warten, bevor man zum nächsten Schritt übergeht. Leider erzielten alle Versuche keine positiven Ergebnisse und aufgrund von Zeitmangel musste letztendlich das Erstellen der Pipeline abgebrochen werden.

Docker

Im Rahmen dieser Übungsaufgabe wurden Dockerfiles erstellt, um sich auf die nächste Aufgabe vorzubereiten. Hierbei wurde ein Tutorial[2] verwendet, um das Grundverständnis für Dockerfiles zu erlangen.

In meinem Projekt sollten Docker-Images für zwei Verzeichnisse erstellt werden, jedoch fehlte noch ein Dockerfile für die MongoDB-Instanz. Da dies meine erste Dockerfile war,

habe ich sie bewusst einfach gehalten. Der Befehl "mongod --bind_ip_all"[3] wurde verwendet, um alle eingehenden IP-Adressen zu akzeptieren.

Die anderen Dockerfiles wurden ähnlich wie der Workflow einer CI-Pipeline aufgebaut, mit dem Unterschied, dass im Frontend mit dem Befehl "npm run build" die Anwendung erstellt wurde. Dadurch kann die Anwendung später im Semester direkt mit dem Docker-Image gestartet werden.

Um alle drei Dockerfiles zusammenzuführen, würde man eine Docker-compose.yml-Datei im obersten Verzeichnis benötigen. Dies ist jedoch Inhalt der späteren Übung, daher wurde sie hier noch nicht geschrieben.

Nachdem die Dockerfiles erstellt wurden, ist es selbstverständlich, sie zu überprüfen. Dafür muss die Docker Desktop App installiert sein und die entsprechenden Docker-Images müssen erstellt werden. Im entsprechenden Terminal verwendet man den Befehl "docker build" mit den entsprechenden Flags, um anzugeben, welche Dockerfiles erstellt werden sollen.

Wenn der Prozess erfolgreich ist, werden die Docker-Images im offiziellen Docker Hub hochgeladen und unter den angegebenen Namen verfügbar sein.

Benachrichtigungen

Beim Einrichten der CI-Pipeline ist mir aufgefallen, dass ich bei jedem Durchlauf der Pipeline mit E-Mails von Github überschwemmt werde. Dies ist für mich sehr unangenehm, da ich bereits durch die heutige Infrastruktur mit Werbungen und anderen E-Mails überlastet bin.

Dies war für mich Anlass genug, die Benachrichtigungen über die Ergebnisse der CI-Pipeline auf eine andere Plattform umzuleiten. Ich habe mich dafür entschieden, den Online-Dienst Discord zu nutzen.

Die Verknüpfung zwischen Discord und Github konnte dank Webhooks sehr schnell hergestellt werden. Die Konfiguration, welche Github-Events eine Benachrichtigung auslösen sollen, erfolgt in den Einstellungen von Github.

Für den Zweck dieser Übung habe ich die Konfiguration so eingestellt, dass nur Workflow-Aktivitäten an Discord gesendet werden. Dadurch erhält man einen besseren Überblick über den Verlauf der Pipeline, anstatt von einer Flut von E-Mails überwältigt zu werden. Ein weiterer Vorteil einer solchen Verknüpfung mit anderen Plattformen besteht darin, dass die Übersichtlichkeit verbessert wird.

Falls später Bedarf besteht, die CI-Pipeline täglich zu starten, könnte man über einen solchen Online-Dienst einen Überblick verschaffen. Ein kleiner Nachteil dieser Funktionalität ist jedoch, dass die Aktivitäten nicht anpassbar sind (obwohl eine Vielzahl von Optionen zur individuellen Auswahl vorhanden ist) und dass für weitere Details letztendlich zu Github weitergeleitet wird.

Natürlich ist Discord nicht die einzige Plattform, die in der Lage ist, Github-Benachrichtigungen weiterzuleiten. Microsoft Teams, das derzeit einer der beliebtesten Kommunikationsdienste für Softwareunternehmen ist, bietet ebenfalls eine solche Verknüpfung an.

In einer professionellen Umgebung können so täglich das gesamte Team über den erfolgreichen Durchlauf der Pipeline informiert werden, ohne dass das Team täglich die Pipeline-Übersicht durchgehen muss. Ein kurzer Blick in den entsprechenden Kanal der bevorzugten Kommunikationsplattform genügt, um sicherzustellen, dass alle erforderlichen Tests erfolgreich abgeschlossen wurden.

Dies spart jedem Teammitglied täglich etwas Zeit, die für produktive Arbeiten genutzt werden kann.

Lessons learned

Das Implementieren einer CI-Pipeline (und später auch einer CD-Pipeline) ist äußerst wichtig für die Qualitätssicherung von Software. Beim Entwerfen der einzelnen Schritte des Workflows ist es jedoch unerlässlich, die Architektur der Software zu verstehen.

Eine Person, die nicht mit dem gesamten Software-Stack vertraut ist, wird Schwierigkeiten haben, eine gut funktionierende CI-Pipeline einzurichten. Daher ist es von entscheidender Bedeutung, neben der Software auch eine qualitativ hochwertige Entwicklerdokumentation bereitzustellen.

Ein weiterer Aspekt, der im Rahmen dieser Übung nicht weiter berücksichtigt wurde, ist die Planung der nächsten Schritte nach dem automatisierten Testen. Mit anderen Worten, der Aspekt der CD-Pipeline wurde noch nicht implementiert. Die bisher eingeführten Schritte sind jedoch ausreichend für das Thema "Continuous Integration".

Außerdem ist mir erneut bewusst geworden, welche Vorteile es mit sich bringt, ordentliche Bash-Skripte schreiben zu können. Bei der Containerisierung kommt Ubuntu bzw. Linux am häufigsten zum Einsatz, und auch in meiner weiteren Karriere als Informatiker sollte ich besser in der Lage sein, eigene Bash-Skripte zu schreiben.

Quellenverzeichnis

[1] Stack Overflow. (2021). Node.js listen EACCES permission denied 0.0.0.0:80. Verfügbar unter:

https://stackoverflow.com/questions/60372618/nodejs-listen-eacces-permission-denied-0-0-0-080

[2] The New Stack. (n.d.). Docker Basics: How to Use Dockerfiles. Verfügbar unter: https://thenewstack.io/docker-basics-how-to-use-dockerfiles/

[3] Stack Overflow. (2019). Difference between mongos bind_ip and bind_ip_all. Verfügbar unter: https://stackoverflow.com/questions/55348228/difference-between-mongos-bind-ip-and-bind-ip-all

Hinweis: Die genannten Quellen dienen als Referenz und wurden zur Unterstützung bei der Lösung spezifischer Probleme und zur Erlangung eines grundlegenden Verständnisses verwendet. Bei der Erstellung dieser Arbeit wurden angemessene Vorkehrungen getroffen, um die Richtigkeit der Informationen sicherzustellen.