Semesterarbeit Aufgabe 3

Matthias Heimberg

Aufgabe 3: JMeter und ZAP

Im Rahmen der Aufgabe 3 soll die Webapplikation mit JMeter und ZAP getestet werden (vgl. Aufgabe 3).

JMeter

Zunächst wird, wie in der Jenkins-Dokumentation beschrieben, das Performance Plugin installiert. ### Docker Image In dieser Arbeit wird, wie bereits in Aufgabe 2 beschrieben, versucht, möglichst alle benötigten Services als Docker Container zu starten. Dazu wird ein Docker Image für JMeter verwendet. Das Docker Image wird von https://hub.docker.com/r/justb4/jmeter bezogen, es ist kein offizielles Docker Image von JMeter. Für ein produktives Setup mit Docker sollte ein eigenes Image erstellt und gewartet werden, dies wird aus Zeitgründen in dieser Arbeit nicht gemacht.

Zugriff auf Docker Socket

Um Docker Container aus Jenkins (welches hier selbst in einem Container läuft) starten zu können, gibt es unterschiedliche Möglichkeiten: - Docker in Docker (DinD) - Zugriff auf Docker Socket (Docker Socket Bind Mount) In https://jpetazzo.github.io/2015/09/03/do-not-use-docker-in-docker-for-ci/ wird aus verschiedenen Gründen (Sicherheit, Kompatibilität) empfohlen, auf Docker in Docker zu verzichten und stattdessen auf den Docker Socket zu binden. Dieser Zugriff ist nicht ganz trivial, da der Jenkins Container nicht als `root` läuft und der Docker Socket nur für `root` zugänglich ist. Dazu wurden folgende Ergänzungen an der `docker-compose.yml` Datei vorgenommen: 1. Ein spezifisches Docker Network wurde erstellt, damit die Container untereinander kommunizieren können (vgl. https://docs.docker.com/network/bridge/) 2. Der Docker Socket wird an den Jenkins Container gebunden (vgl. https://docs.docker.com/engine/security/rootless/)

Damit sieht die `docker-compose.yml` Datei wie folgt aus:

```yaml
version: "3.8"
services:
  jenkins:
      image: jenkins/jenkins:lts
      container_name: jenkins
      user: root
      ports:
          - "8080:8080"
          - "50000:50000"
      volumes:
          - ./jenkins-data:/var/jenkins_home
          - //var/run/docker.sock://var/run/docker.sock
      restart: always
      networks:
          - jenkins

  sonarqube:
      image: sonarqube:latest
      container_name: sonarqube
      ports:
          - "9000:9000"
          - "9092:9092"
      volumes:
          - ./sonarqube-data:/opt/sonarqube/data
          - ./sonarqube-extensions:/opt/sonarqube/extensions
          - ./sonarqube-logs:/opt/sonarqube/logs
          - ./sonarqube-temp:/opt/sonarqube/temp
      restart: always
      networks:
          - jenkins

networks:
    jenkins:
        driver: bridge
```

Wird `docker compose` auf einem Windows Host ausgeführt, muss zunächst die Variable `COMPOSE_CONVERT_WINDOWS_PATHS=1` gesetzt werden, damit der Pfad korrekt gemountet wird. Hierzu wird eine `.env` Datei erstellt, welche folgenden Inhalt hat:

```
COMPOSE_CONVERT_WINDOWS_PATHS=1
```

Diese Datei muss im selben Verzeichnis wie die `docker-compose.yml` Datei liegen. Zudem muss Docker im Jenkins Container installiert werden. Dies erfolgt manuell

in der CLI mittels `curl https://get.docker.com/ > dockerinstall && chmod 777 dockerinstall && ./dockerinstall`.

Ist Docker im Jenkins Container installiert, kann nun aus dem Jenkins Container auf den Docker Socket zugegriffen werden. Damit lässt sich JMeter in einem Docker Container parallel zum Jenkins Container starten und ausführen. Die Test-Scripte müssen im Verzeichnis `/jmeter-data/scripts` liegen, damit sie vom Jenkins Container aus gefunden werden können.

Einbinden von JMeter in das Jenkinsfile

Die JMeter Tests werden als weiterer Step im Jenkinsfile definiert. Dazu wird der folgende Code verwendet:

```
// test with jmeter inside docker container (jenkins container binds to
// docker socket on host)
        stage('test with jmeter') {
            steps {
                gitlabCommitStatus(name: 'test with jmeter') {
                    sh '''
                        export TIMESTAMP=$(date +%Y%m%d_%H%M%S)
                        export JMETER_PATH=/mnt/jmeter
                        docker run --rm -v jmeter-data:"${JMETER_PATH}" /
                        justb4/jmeter -n -t /mnt/jmeter/scripts /
                        -l "${JMETER_PATH}"/tmp/result_"${TIMESTAMP}".jtl /
                        -j "${JMETER_PATH}/tmp/jmeter_${TIMESTAMP}".log
                    '''
                }
            }
        }
```

Der Befehl `docker run` startet einen Docker Container mit dem Image `justb4/jmeter`. Dieses Image enthält JMeter (Quelle: github.com/justb4/docker-jmeter). Der Befehl `--rm` sorgt dafür, dass der Container nach dem Test gelöscht wird. Das Volume `jmeter-data` wird an den Pfad `/mnt/jmeter` gemountet. Der Pfad `/mnt/jmeter` ist der Pfad, an dem JMeter im Container erwartet, dass die Test-Scripts liegen. Der Befehl `-n` sorgt dafür, dass JMeter im Non-GUI Modus läuft. Der Befehl `-t` gibt den Pfad an, an dem die Test-Scripts liegen. Der Befehl `-l` gibt den Pfad an, an dem das Ergebnis des Tests gespeichert werden soll. Der Befehl `-j` gibt den Pfad an, an dem das Log des Tests gespeichert werden soll.

Fehlschlag beim Einbinden des Volumes

Probleme mit dem Einbinden des Volumes `jemeter-data` vom Host in den JMeter Container, welcher über den Docker Socket des Hosts aus dem Jenkins Container heraus gestartet wird, konnten auch nach mehreren Stunden nicht gelöst werden. Deshalb wurde nach einiger Recherche der folgende Ansatz verfolgt: 1. Jmeter wird wie zuvor in einem Docker Container gestartet 2. Der Start des JMeter Containers wird durch einen `docker agent` im Jenkinsfile ausgeführt 3. Dazu müssen die Plugins `Docker` und `Docker Pipeline` in Jenkins installiert werden

Damit wird der entsprechende Step im Jenkinsfile wie folgt abgeändert:

```
stage('test with jmeter') {
            agent {
                docker {
                    image 'justb4/jmeter:5.1.1'
                    args '-v /var/jenkins_home/jmeter-data:/home/user/jmeter'
                }
            }
            steps {
                gitlabCommitStatus(name: 'test with jmeter') {
                    sh '''
                        export TIMESTAMP=$(date +%Y%m%d_%H%M%S)
                        jmeter -n -t /home/user/jmeter/check_api.jmx /
                        -l /home/user/jmeter/result_${TIMESTAMP}.jtl /
                        -j /home/user/jmeter/jmeter_${TIMESTAMP}.log
                    '''
                }
            }
        }
```

Das Skript kann aber wieder nicht gestartet werden, da der Entrypoint des Containers nicht dem vom Agent erwarteten Entrypoint entspricht. Der folgende Fehler wird ausgegeben:

```
ERROR: The container started but didn t run the expected command.
Please double check your ENTRYPOINT does execute the command passed
as docker run argument, as required by official docker images
```

Wird der Entrypoint auf `/bin/sh` gesetzt, so wird der folgende Fehler ausgegeben:

```
java.io.IOException: Failed to run top
c962a508d1a0b0c9fda3eb71f65228e48a61fac9ce9511907fb0a40561eedc44.
Error: Error response from daemon: Container
```

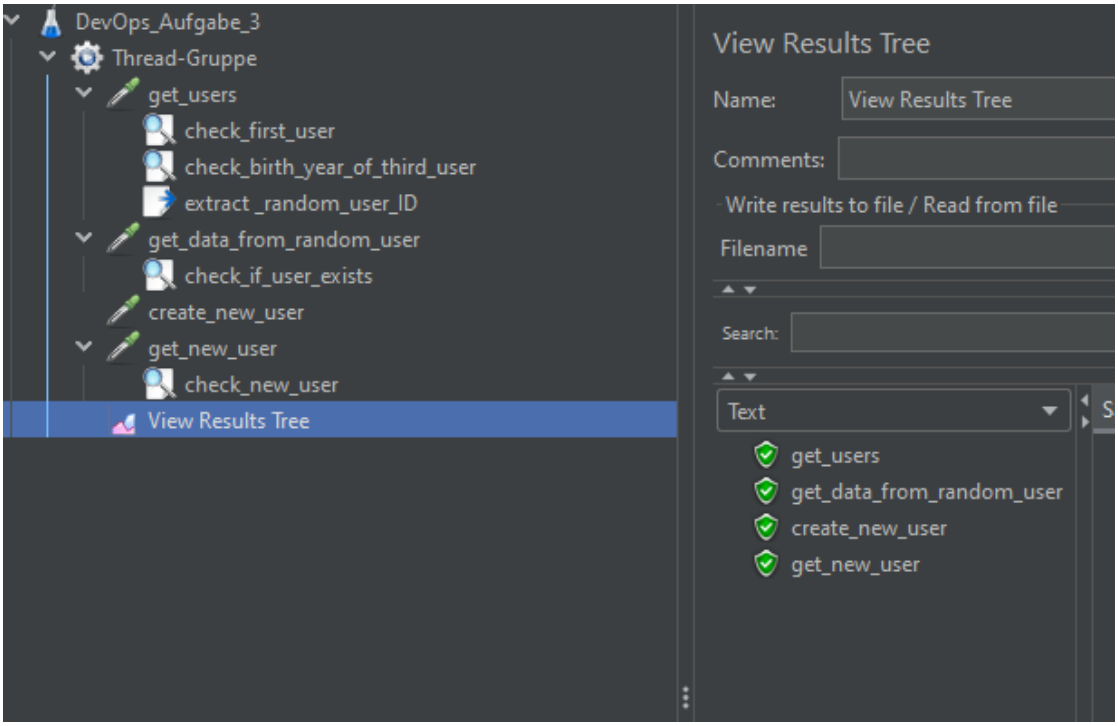
```
c962a508d1a0b0c9fda3eb71f65228e48a61fac9ce9511907fb0a40561eedc44
is not running
```

Obwohl der Container nachweislich gestartet wird (er wird aber kurz nach Start wieder beendet, weshalb Jenkins das Kommando `docker ps` nicht ausführen kann). Auf Grund der zeitlichen Beschränkungen wurde der Ansatz dann nicht mehr weiter verfolgt und der entsprechende Step im Jenkinsfile auskommentiert.

JMeter Test-Scripts

Die Test Scripts wurden mit einer lokalen Installation von JMeter erstellt und als `.jmx` Dateien exportiert. Dabei wurden die folgenden Tests definiert:

1. `get_users`: Holt mittels `GET` alle Users vom Endpunkt `/api/users`
    1. `check_first_user`: Prüft, ob der erste User den Namen `Bruno` besitzt
    2. `check_birth_year_of_third_user`: Prüft, ob das Geburtsjahr des dritten Users `2001` ist
    3. `extract _random_user_ID`: Extrahiert eine zufällige User ID und weist diese der Variable `${id}` zu
2. `get_data_from_random_user`: Holt mittels `GET` die Daten des Users mit der ID `${id}` vom Endpunkt `/api/users/${id}`
    1. `check_if_user_exists`: Prüft, ob der User mit der ID `${id}` existiert indem geprüft wird, ob dieser einen Namen besitzt
3. `create_new_user`: Erstellt mittels `POST` einen neuen User auf dem Endpunkt `/api/users` mit den Daten `{"id": 6, "name": "Hans", "email": "hans@hans.com", "birthYear": 1990}`
4. `get_new_user`: Holt mittels `GET` den neu erstellten User vom Endpunkt `/api/users/6`
    1. `check_new_user`: Prüft, ob der neu erstellte User existiert indem geprüft wird, ob dieser den Namen `Hans` besitzt



Probleme und deren Lösung

- `docker compose` lässt sich nicht starten, da ein Port bereits gelegt ist. Lösung: belegte Ports lassen sich unter Windows mittels des PoweShell-Befehls `Get-Process -Id (Get-NetTCPConnection -LocalPort <PORT>).OwningProcess` herausfinden. Anschliessend kann der entsprechende Prozess beendet werden.
- Das `performance plugin` steht im Jenkins-GUI nicht zur Verfügung. Das Plugin wird innerhalb des laufenden Jenkins Containers manuell via über den CLI Befehl `jenkins-plugin-cli --plugins performance:3.20` installiert.
- JMeter lässt sich nicht aus dem Jenkins Container starten. Lösung: Es wird ein `docker agent` im Jenkinsfile definiert. Die JMeter Befehle können dann im entsprechenden Step deklariert werden. Der `docker agent` startet einen Docker Container, in dem JMeter ausgeführt wird.
- Jenkins erkennt den `docker agent` nicht. Lösung: Das Plugin `docker pipeline` in Jenkins installieren.