Verteiltes System Protokoll
Trung Thieu Quang – 771043

1. UDP Socket

- 2 Komponenten: Bank in bank.mjs und Börse in boerse.mjs file, außerdem gibt es noch eine Klasse Wertpapier in wertpapier.mjs
- In Banken.mjs, UDP Server wird durch Funktion startServer() implementiert

```
startServer() {
    //create a new udp socket (udp4 = ipv4)
    const server = dgram.createSocket('udp4');
    //event listener for message , emit when a new datagram is available on socket
    //msg: buffer containing the incoming message
    //rinfo: containing sender's address, port, size of datagram
    server.on('message', (msg, rinfo) => {
        // console.log(`Received data: ${msg.toString()}`);
        const parsedData = JSON.parse(msg.toString());
        this.receiveData(parsedData.wertpapier, parsedData.count, parsedData.preis);
        const responseBuffer = Buffer.from(`Received from Boerse: ${rinfo.address}, on port ${rinfo.port} ${parsedData.wertpapier.kurzel}, ${parsedData.count}`);
        //send a reponse back to client
        server.send(responseBuffer, rinfo.port, rinfo.address, (err) => {
            if (err) {
                console.log('Error sending response:', err);
            } else {
                console.log("sent message to client");
            }
        });
    });
    //emit when the server is on (bound to an address and port, ready for listening)
    server.on('listening', () => {
        const address = server.address();
        console.log(`Bank server listening on ${address.address}:${address.port}`);
    });
    //server start listening for incoming data gram ipAddress:port
    server.bind(this.port);
}
```

Hier in der Funktion, `const server = dgram.createSocket('udp4');` erstellt einen neuen Datagram-Socket mit UDP

- Dieser Prozess findet auch bei Boerse statt
- Dann verbinde ich Boerse und Bank mit der Funktion connectToBank(bank) in Boerse.mjs

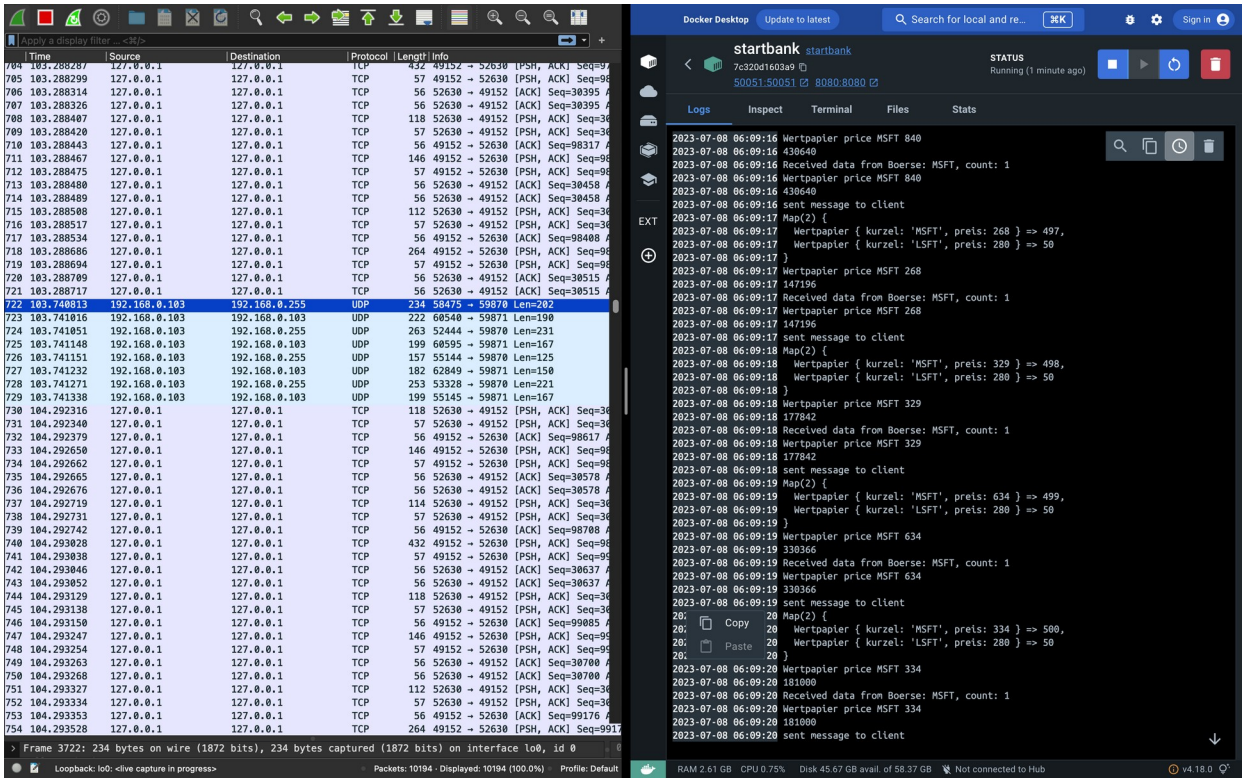
```
connectToBank(bank) {
    this.client = dgram.createSocket('udp4');
    this.connectedBank = bank;
    console.log(`connected to bank ${bank.name} at ${bank.ipAddress} on port ${bank.port}`);
    this.client.on('message', (msg, rinfo) => {
        console.log(msg.toString());
    });
}
```

- Die gesendete Nachrichten werden durch server.on(‘message, (msg, rinfo) => … in startServer gehört und gehandelt

- Die von Boerse gesendete Daten sind in JSON Format, hier ist der Preis zufällig

```
setInterval(() => {
    var price = Math.floor(Math.random() * 1000);
    var amount = Math.floor(Math.random()* 1000);
    MSFT.updatePrice(price)
    b1.addWerpapier(MSFT, 1);
    console.log(b1.wertpapiers);
    try {
        b1.sendData(MSFT, 1);
    } catch(e) {
        console.log(e);
    }
}, 1000);
```

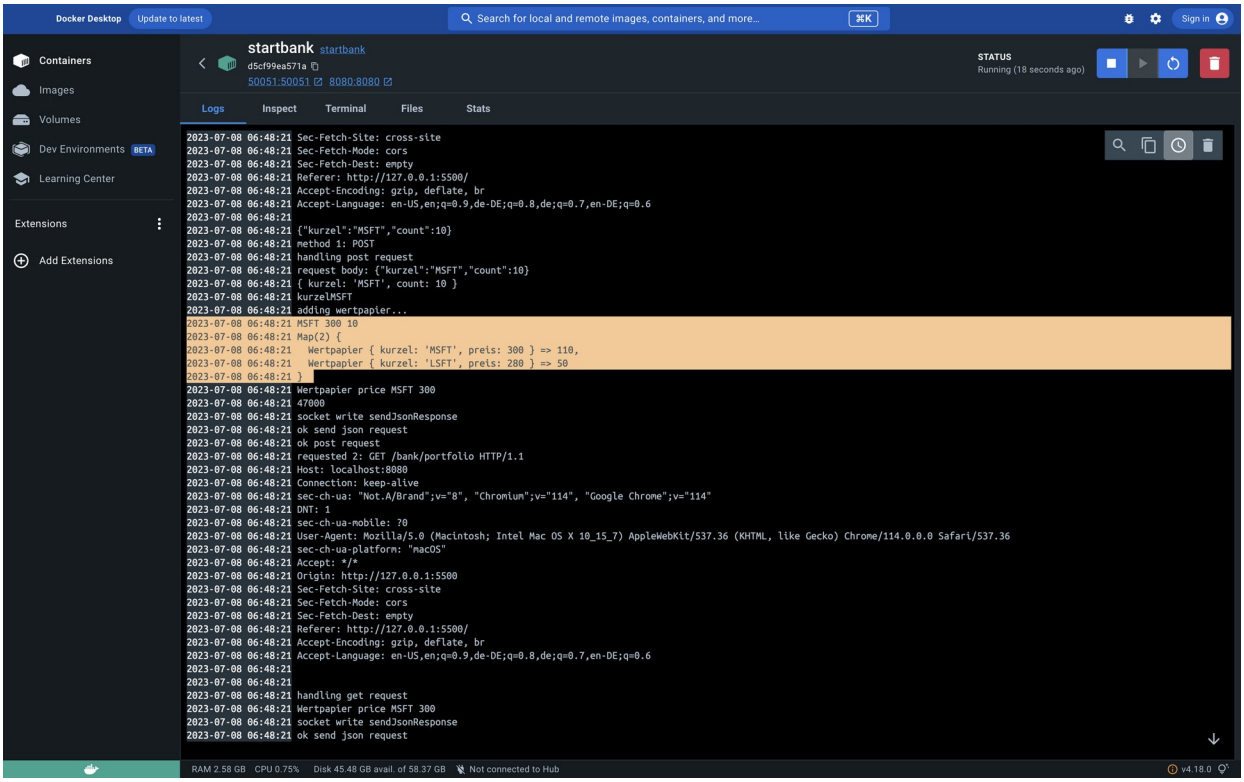
Für den Performanz-Test, hier muss ich keine Code schreiben, Ich laufe hier beide Bank und Boerse in docker mit –network=host, damit kann ich RTT mit Wireshark messen



- Hier kann ich z.B die Pakets, die zwischen Bank und Boerse gesendet werden, sehen
- Der Zeitunterschied zwischen den beiden Paketen beträgt lediglich etwa 0,00203s oder 2,03ms, was zu erwarten ist, da sie beide auf demselben Rechner (localhost) ausgeführt werden.

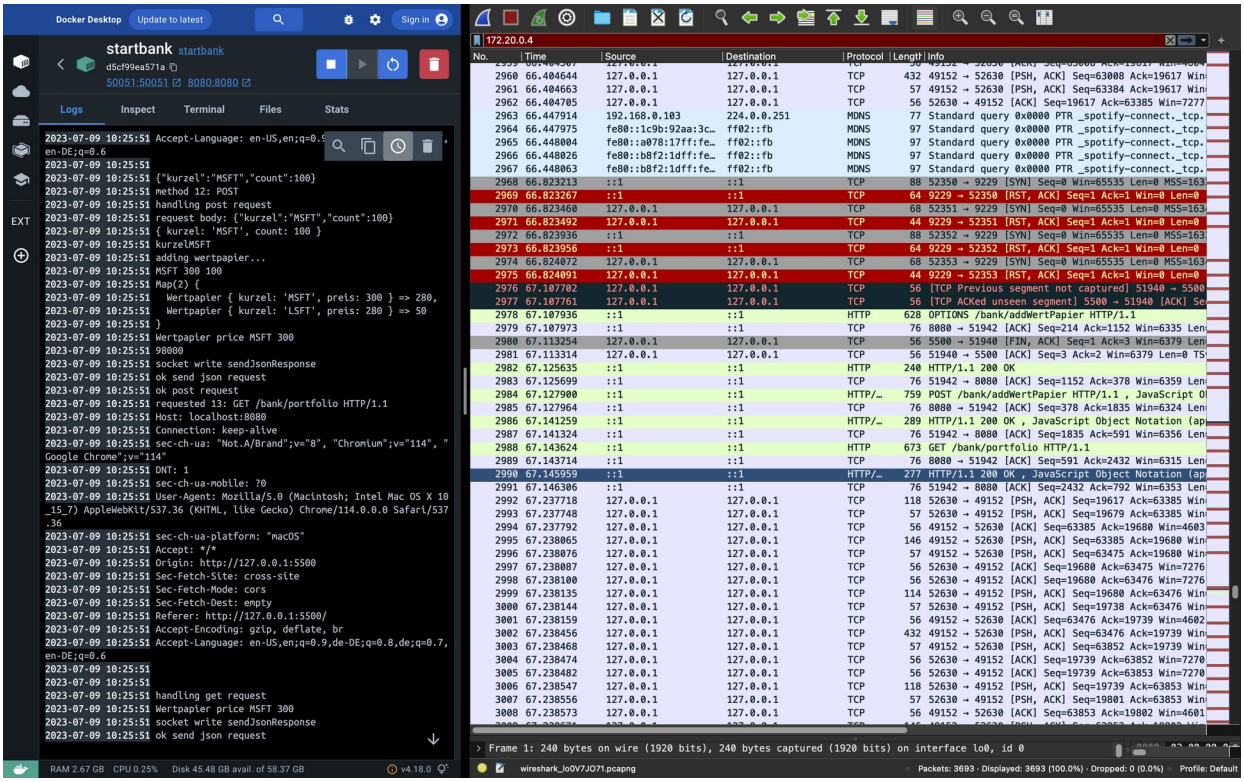
2. TCP

- Die Funktion startHttpServer() startet einen HTTP-Server unter Verwendung des Node.js-Netzwerkmoduls (net module)
  - net.createServer erstellt einen neuen TCP-Server.
  - Der Server überprüft, ob er alle HTTP-Header vom Client erhalten hat. In dem HTTP-Protokoll werden die Header zuerst gesendet und vom Body der Anfrage durch eine leere Zeile getrennt, die durch \r\n\r\n im String repräsentiert wird. Wenn das Ende der Header gefunden wurde, fährt der Server mit der Verarbeitung fort.
  - Die HTTP-Header werden in Zeilen aufgeteilt, und die erste Zeile, die die HTTP-Methode (wie GET, POST, OPTIONS usw.) und den Anfragepfad enthält, wird extrahiert. Der Server versucht auch, den Content-Length-Header zu finden, der angibt, wie viele Bytes an Body-Daten nach den Headern erwartet werden.
  - Wenn ein Content-Length-Header gefunden wurde, überprüft der Server, ob er bereits alle Body-Daten erhalten hat. Wenn dies der Fall ist, bearbeitet der Server die Anfrage entsprechend ihrer Methode. Wenn der Content-Length-Header nicht gefunden wurde, geht der Server davon aus, dass die Anfrage keinen Body hat, und bearbeitet die Anfrage entsprechend ihrer Methode.
  - Nachdem die Anfrage bearbeitet wurde, wird requestData für die nächste Anfrage gelöscht. In HTTP/1.1 kann dieselbe TCP-Verbindung für mehrere Anfragen/Antworten verwendet werden (dies wird als "keep-alive" bezeichnet), daher muss der Server bereit sein, die nächste Anfrage auf derselben Verbindung zu bearbeiten.
  - Response werden durch Methoden wie sendJsonResponse, sendResponse, … am ende der Bank-Klasse mit Hilfe von socket.write() gesendet
- Bank Interface wird alle in index.html geschrieben, dort wird UI für Aufgabe 2, 3 und 4 implementiert, deshalb kann man die Funktionalitäten einfacher testen
- Hier ist ein Beispiel wenn ich 10 MSFT durch Browser sende



- DSDSD

- RTT Messen: Start Paket: 2978, End Paket: 2990, RTT ca. 38Ms (Hier sendet Browser unter Anwendung von Funktion addWertpapier 3 Anfrage, 2 Anfrage auf Portfolio und eine auf Methode addWertpapier()



3. RPC

- Hier wird gRPC implementiert
- Zuerst muss ich Message und Service in bank.proto definieren, da IDL von gRPC Protocol Buffer ist

- Dann definiere ich Path zu bank.proto in bank.mjs mit PROTO_PATH und lade Protobuf Definitions in packageDefinition

```
26  const PROTO_PATH = 'bankService.proto';
27
28  const packageDefinition = pkg.loadSync(
29      PROTO_PATH,
30      {
31          keepCase: true,
32          longs: String,
33          enums: String,
34          defaults: true,
35          oneofs: true
36      }
37  );
```

- Da ein Bank als Client und Server funktioniert, in dem Konstruktor werden rpcClientPort und ServerPort festgestellt

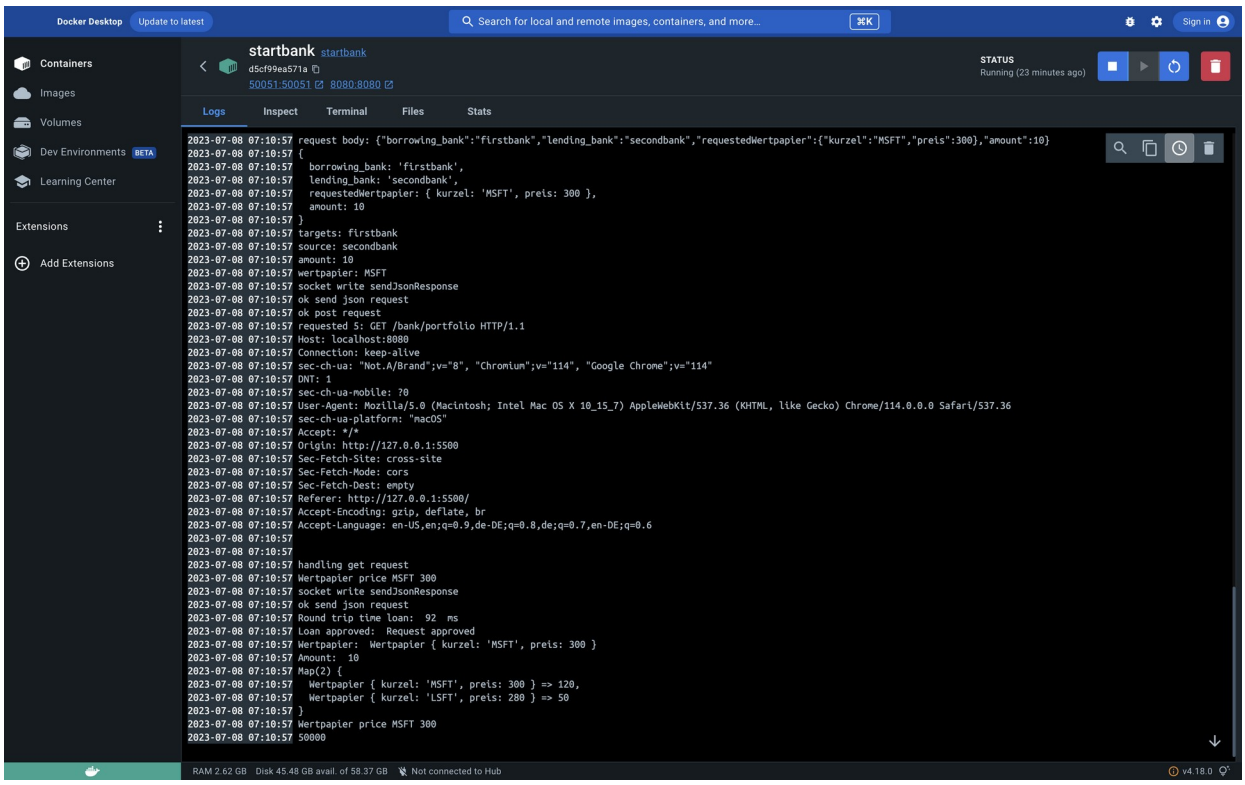
```
60      //Server port for incoming data
61      this.rpcServerPort = rpcServerPort;
62
63      //client port for outgoing data
64      this.rpcClientPort = rpcClientPort;
65      this.server = new grpc.Server();
66
67      //Service set for the bank
68      this.server.addService(bank_proto.service, {
69          requestLoan: this.requestLoan.bind(this), //binding bank object to requestLoan function, so that we can use class method later
70          transferFunds: this.transferFunds.bind(this)
71      });
72
73      //list of client to send grpc request
74      this.clients = new Map();
75      this.loanList = new Map();
76  }
```

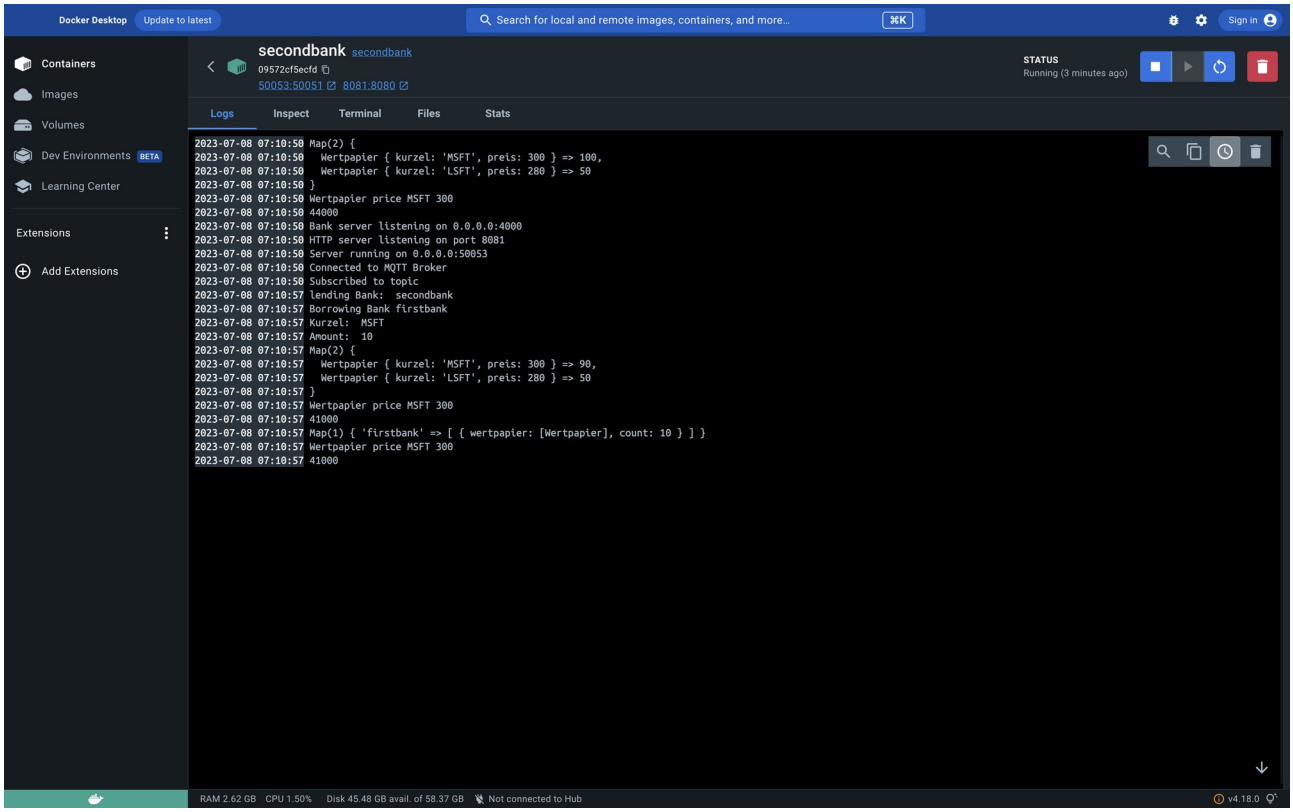
- Die Methode addService wird verwendet, um einen Dienst zu Ihrem gRPC-Server hinzuzufügen
- Hier erstelle ich auch ein client Map und loanList map, um Kreditverlauf zu verfolgen
- RPC Server wird hier gestartet

```
211  startGRPC() {
212      this.server.bindAsync(`0.0.0.0:${this.rpcServerPort}`, grpc.ServerCredentials.createInsecure(), () => {
213          this.server.start();
214          console.log(`Server running on 0.0.0.0:${this.rpcServerPort}`);
215      });
216  }      You, 3 weeks ago • update aufgabe 3
```

- Die zwei Methode requestLoan und transferFund sind Implementationen von Service, die in bankService.proto definiert werden
- Diese beiden Funktionen, makeTransferRequest und makeLoanRequest, sind clientseitige Methoden zum RPC, die vom BankService bereitgestellt werden.

- Hier ist ein Testfall, indem wir Kreditsanfrage zwischen firstBank und secondBank testen

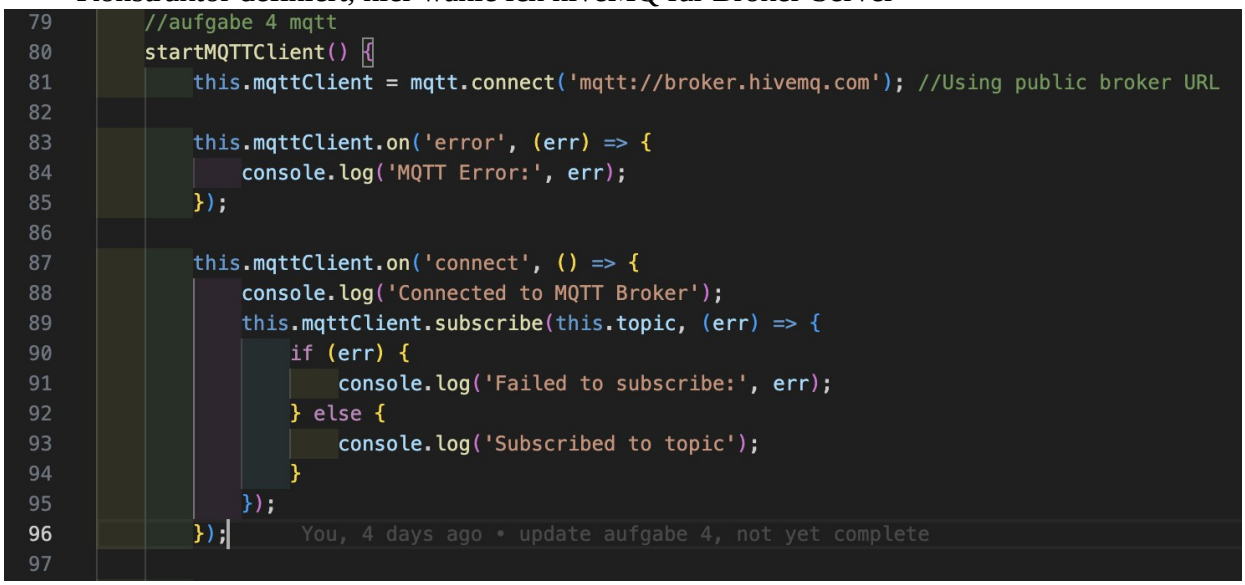




- RTT für Kreditsprozess beträgt hier 23-92ms (in erstem Bild gezeigt)
  Durchschnitt von 10 Anfragen: ca. 45ms

4. MOM – MQTT

- Die Funktion startMQTTClient verbindet sich mit dem MQTT-Broker. Danach abonniert sie ein bestimmtes Topic und setzt einen Listener für eingehende Nachrichten. Topic wird in Konstruktor definiert, hier wähle ich hiveMQ für Broker-Server



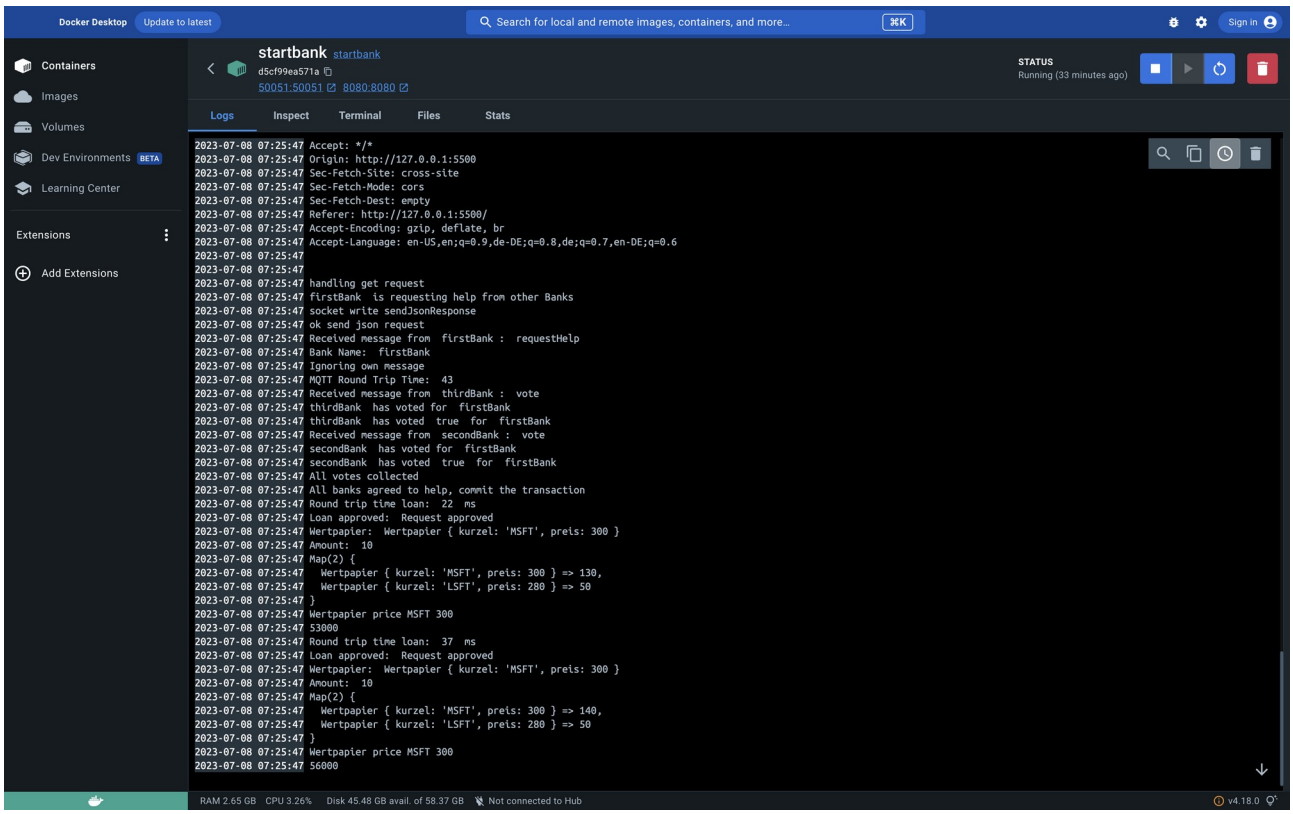
```
79      //aufgabe 4 mqtt
80      startMQTTClient() {
81          this.mqttClient = mqtt.connect('mqtt://broker.hivemq.com'); //Using public broker URL
82
83          this.mqttClient.on('error', (err) => {
84              console.log('MQTT Error:', err);
85          });
86
87          this.mqttClient.on('connect', () => {
88              console.log('Connected to MQTT Broker');
89              this.mqttClient.subscribe(this.topic, (err) => {
90                  if (err) {
91                      console.log('Failed to subscribe:', err);
92                  } else {
93                      console.log('Subscribed to topic');
94                  }
95              });
96          });      You, 4 days ago • update aufgabe 4, not yet complete
97
```

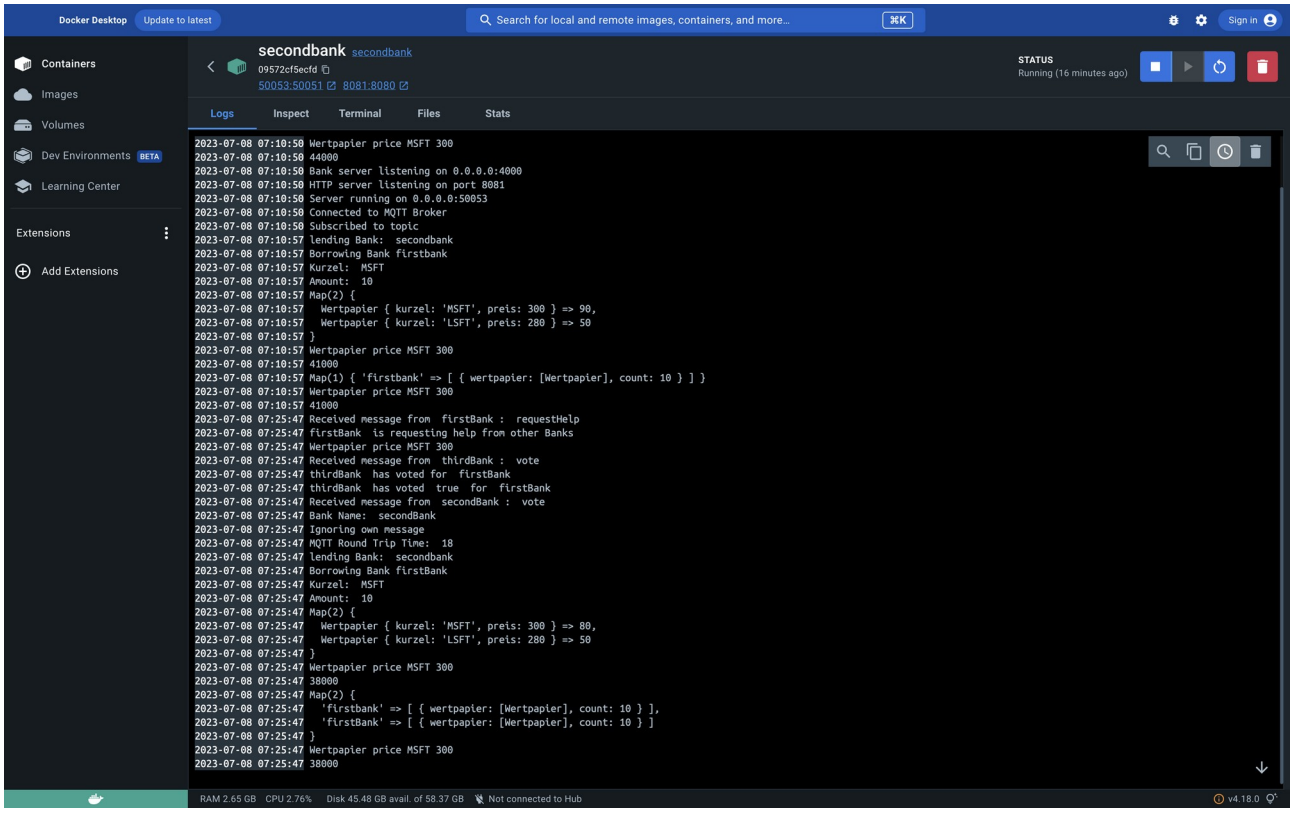
- Hilfeanforderung: Wenn eine Bank Unterstützung benötigt, ruft sie die Funktion requestHelp auf. Diese Funktion markiert die Bank als Hilfe suchend und sendet eine Nachricht vom Typ requestHelp an alle anderen Banken über das MQTT-Topic.

- 2PC Algorithmus für Abstimmungsprozess:
  - Abstimmung: Bei Empfang einer requestHelp-Nachricht von einer anderen Bank reagiert die Bank mit einer Abstimmung. Dies geschieht in der Funktion vote. Die Abstimmung wird basierend auf dem Zustand des Portfolios der Bank entschieden. Die Abstimmung wird dann als vote-Nachricht an das Topic gesendet.
  - Abstimmungen sammeln: Bei Empfang einer vote-Nachricht von einer anderen Bank wird diese in der Funktion collectVote gesammelt. Es wird überprüft, ob die Anforderung von der Bank selbst stammt oder ob die Bank aktuell Hilfe benötigt. Wenn alle Stimmen gesammelt sind, wird entschieden, ob der Hilfsprozess fortgesetzt oder abgebrochen wird.
  - Hilfsprozess fortsetzen oder abbrechen: Wenn alle Banken zugestimmt haben, den Hilfsprozess fortzusetzen, wird die Funktion commitHelp aufgerufen. Hier werden Darlehensanfragen an alle anderen Banken gesendet. Falls eine der Banken gegen den Hilfsprozess stimmt, wird die Funktion abortHelp aufgerufen und der Prozess wird abgebrochen.
- Berechnung der Round-Trip-Zeit: Um die Leistung des Systems zu bewerten, wird die Round-Trip-Zeit (RTT) für MQTT-Nachrichten berechnet und ausgegeben. Dies geschieht in der Funktion calculateRTT.

```
136     calculateRTT(startTime, endTime) {
137         return endTime - startTime;
138     }
139     sendMqttMessage(topic = this.topic, messageObject) {
140         this.startMqttTime = new Date();
141         const messageString = JSON.stringify(messageObject);
142         this.mqttClient.publish(topic, messageString);
143     }
144
145     requestHelp() {
146         console.log(this.name, ' is requesting help from other Banks');
147         this.needHelp = true;
148         this.votes = new Map();  // To collect the votes
149         this.sendMqttMessage(this.topic, {
150             type: 'requestHelp',
151             sender: this.name
152         });
153     }
154
155     vote(requestingBank) {
156         const vote = this.calculatePortfolio() > 30000; // Decide the vote based on some condition
157         this.sendMqttMessage(this.topic, {
158             type: 'vote',
159             sender: this.name,
160             requester: requestingBank,
161             vote
162         });
163     }
164
165     collectVote(votingBank, vote, requestingBank) {
166         console.log(votingBank, ' has voted ', vote,' for ', requestingBank);
167         if (!this.needHelp && requestingBank !== this.name) {
168             return;
169         }
170         this.votes.set(votingBank, vote);
171         // Check if all votes are collected
172         if (this.votes.size === this.clients.size) {
173             console.log('All votes collected');
```

MQTT TestFall:





MQTT RTT beträgt 18ms-60ms (10 Anfragen)