Der Programmcode ist in dem folgenden GitHub Repository zu finden: `https://github.com/s4paweil/SA4E_Ueb03.git`
Eine Anleitung zum Ausführen des Programms ist der `README.md` zu entnehmen.

Im Folgenden wird die Lösung für das 3. Übungsblatt - die Implementierung einer Simulation des Spiels Ave Caesar unter Verwendung einer Streaming-Architektur - vorgestellt. Hierbei werden nicht die einzelnen Aufgaben, sondern direkt das vollständige Ergebnis präsentiert.
Zunächst wird - um dem Namen der Veranstaltung gerecht zu werden - die zugrundeliegende Softwarearchitektur vorgestellt, bevor anschließend der Programmablauf genauer erklärt wird. Anschließend werden nochmal explizit die implementierten Regeln des Ave Caesar Spiels aufgezeigt, bevor in einem Fazit die Bearbeitung der Übung zusammengefasst wird.

1 Softwarearchitektur

Als Streaming-Architektur wurde sich für Apache Kafka, ein Event-Streaming Framework, entschieden, da es in der Praxis weit verbreitet ist und eine hohe Relevanz besitzt. Die gesamte Spiele-Simulation läuft in der Docker Umgebung ab. Hierbei sind zum einen die 3 Kafka-Instanzen, die zusammen das Kafka-Cluster bilden, und zum anderen die einzelnen Segmente jeweils als eigene Container implementiert. Das Kafka-Cluster wird mit Hilfe von *KRaft* und somit explizit ohne *Zookeeper* orchestriert. Abbildung 1 zeigt einen groben Überblick über die Architektur.

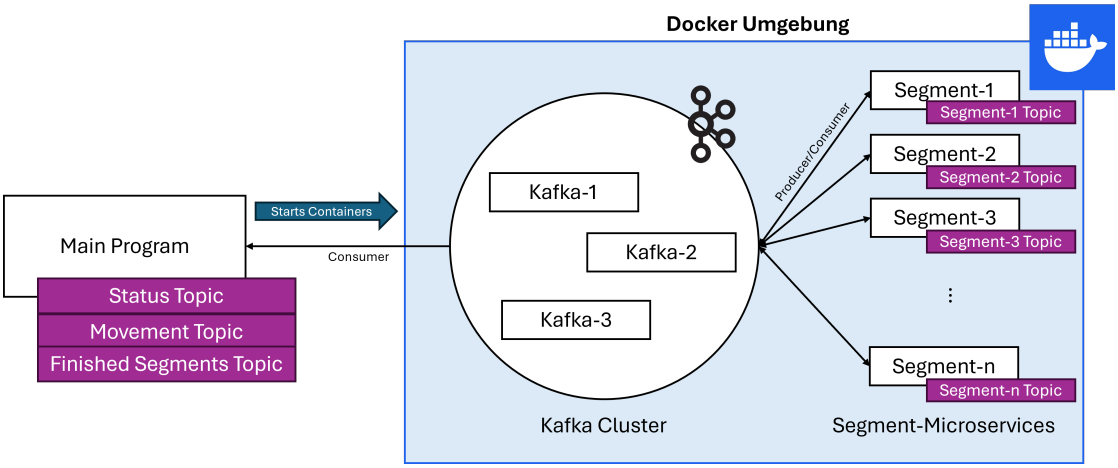


Abbildung 1: Softwarearchitektur der Ave Caesar Simulation.

Über das Hauptprogramm werden mit Hilfe eines `docker-compose.yaml`-Files alle erforderlichen Container gestartet. Die gesamte Kommunikation, sowohl zwischen den Segment-Microservices als auch die Überwachung der Bewegungen durch das Hauptprogramm, erfolgen über Event-Streams mit Kafka. Hierzu ist jeder Segment-Microservice sowohl

als Comsumer, als auch Producer, tätig. Für die Kommunikation der Segmente untereinander lauscht jedes Segment als Consumer einem eigenen Topic, das dem Namen des Segments entspricht. Dadurch können die Segmente beliebig Nachrichten untereinander austauschen, vorausgesetzt der Segment-Name ist bekannt. Da jedes Segment die Namen seiner Nachfolger kennt (`nextSegments`) kann so das Spielfeld abgelaufen werden. Eine genauere Beschreibung des Programmablaufs folgt im nächsten Abschnitt.
Macht eine Spielfigur, also ein Streitwagen, einen Zug, wird dieser via dem Topic `MOVEMENTS_TOPIC` an das Hauptprogramm gesendet. Dadurch kann das Hauptprogramm die Züge der Spieler ausgeben.

2 Programmablauf

Im Kern besteht die Simulation aus drei Python Files: Dem Hauptprogramm `run.py`, dem Setup-Programm zum Erstellen der Konfigurationen `setup.py` und dem Code für die Segment- Microservices `segment.py`.

Die Spiele Simulation läuft wie folgt ab:

1. **Setup:** Zum Starten der Simulation führt der Nutzer das Hauptprogramm **run.py** aus. Es werden zwei Argumente übergeben: Die Anzahl sowie die Länge der Tracks. Im ersten Schritt führt das Hauptprogramm `setup.py` aus, wobei die beiden Argumente mit übergeben werden. `setup.py` erstellt basierend darauf eine `map.json`, die die Struktur des Spielfelds mit all seinen Segmenten enthält. Anschließend wird basierend auf dem Layout des Spielfelds ein `docker-compose.yaml` File erzeugt, in dem sowohl die drei Kafka Container als auch für jedes Segment ein Container enthalten sind. Wichtig ist bereits in dem Compose-File die Startreihenfolge der Container durch Bedingungen zu berücksichtigen, da zuerst die Kafka Infrastruktur laufen soll, bevor die Segment-Container gestartet werden.

2. **Startvorgang:** Sobald das Docker-Compose-File zur Verfügung steht, beginnt das Hauptprogramm mit dem Startvorgang. Es führt das Compose-File aus, wodurch alle Container - beginnend mit der Kafka-Infrastruktur - gestartet werden. Anschließend wartet das Hauptprogramm darauf, dass alle Container korrekt ausgeführt werden.
   Nach dem Start senden alle Segment-Container eine *"I am Alive"* Nachricht seafile `STATUS_TOPIC` via Kafka an den Hauptprozess. Sobald dieser von jedem Container eine entsprechende Nachricht erhalten hat, sendet er eine *SStart"*-Nachricht an die Start-Ziel-Segmente, woraufhin die Simulation beginnt.
   Im weiteren Verlauf lauscht der Hauptprozess auf dem `MOVEMENT-TOPIC`, über das die Container die Bewegungen der Spieler übermitteln. Diese gibt der Hauptprozess dann auf der Konsole aus. Abbildung 2 zeigt beispielhaft eine Ausgabe für einen Simulationsdurchlauf mit 2 Spielern.

3. **Simulation:** Alles Segmente lauschen als Consumer jeweils einem eigenen Topic, das zur Vereinfachung dem Namen des jeweiligen Containers entspricht. Erhält ein Segment eine *SStart"*-Nachricht oder Bewegt sich ein Spieler auf ein neues Feld (das entspricht einer *"Move"*-Nachricht), startet das Segment einen Scouting-Prozess mit dem Ziel, das neue beste Feld basierend auf den drei verfügbaren Karten des Spielers zu finden. Da jedes Segment nur die eigenen Folgesegmente kennt, startet eine Tiefensuche, wobei jedes Segment die *SScouting"*-Nachricht an seine Nachfolger weiterleitet. Am Ende der Tiefensuche erhält das Ursprungssegment eine Liste verfügbarer Ziele und wählt daraus die beste Option aus. Bevorzugt werden - falls Caesar noch nicht gegrüßt wurde - Segmente vom Typ Caesar, ansonsten wird das Segmente ausgewählt, für das die größte Karte verwendet wird - also das Segment, bei dem man am weitesten läuft.

Beim Überqueren eines Start-Ziel-Segments wird die Anzahl der zurückgelegten Runden entsprechend erhöht. Nach einer festgelegten Anzahl von Runden (standardmäßig 3) wird das Spieler Token nicht mehr weitergeleitet, heißt der Spieler hat das Ziel erreicht.

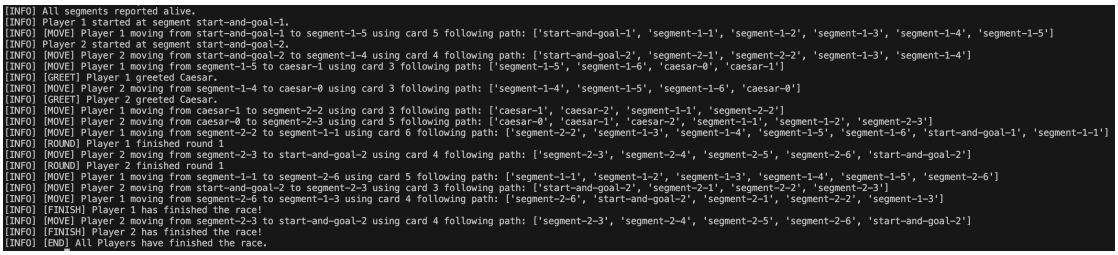
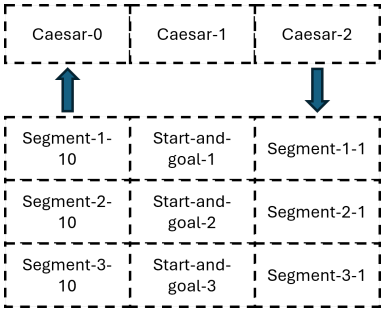


Abbildung 2: Beispielausgabe für einen Simulationsdurchlauf mit 2 Spielern.

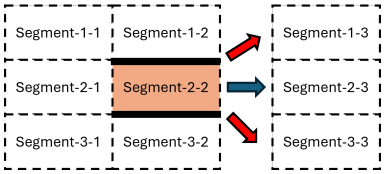
3 Implementierte Regeln des Spiels

Bei der vorliegenden Implementierung des Ave-Caesar Spiels handelt es sich um eine vereinfachte Simulation des Brettspiels. Dennoch wurden einige Regeln des Spiels berücksichtigt und integriert. Folgende Regeln wurden in die Simulation übernommen. Abbildung 3 zeigt eine visuelle Darstellung einiger der genannten Regeln:

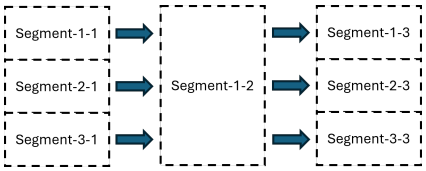
- **Kaisergasse:** Bei der Generierung des Spielfelds wird parallel zu den Start-Ziel-Segmenten auch sogenannte Kaisergasse erstellt. Jeder Spieler muss, um das Spiel zu gewinnen, vor Abschluss der letzten Runde einmal die Kaisergasse befahren und dort angehalten haben.
- **Engpässe/Verbreiterungen:** Bei der Generierung des Spielfelds werden auch Engpässe integriert, sodass zwei Segmente zu einem folgenden Segment zusammengeführt werden. Ebenso gibt es auch Verbreiterungen, bei denen aus einem Segment zwei Folgesegmente resultieren. Die maximale Anzahl paralleler Segmente wird dabei allerdings nicht überschritten, es kann also nicht beliebig viele Verbreiterungen geben.
- **Fahrbahnbeschränkungen:** Bei der Generierung des Spielfelds werden auch seitliche Beschränkungen der Fahrbahn berücksichtigt, sodass Spieler nicht die Fahrbahnen (Tracks) wechseln können. Auf dem originalen Spielfeld sind diese Beschränkungen visuell durch Zäune oder Mauern dargestellt, die nicht überfahren werden dürfen.
- **Rennkarten:** Anstelle des klassischen Würfels existieren bei Ave Caesar sogenannte Rennkarten. Jeder Spieler hat immer 3 Rennkarten auf der Hand, diese haben Werte zwischen 1 - 6. Ist ein Spieler am Zug kann er wählen, welche seiner Karten er spielen möchte und die entsprechende Distanz auf dem Spielfeld zurückzulegen.



(a) Caesar-Gasse oberhalb der ersten Tracks, parallel zum Start/Ziel.



(b) Begrenzung durch Wände, nur horizontale Bewegung erlaubt.



(c) Bei Engpässen existiert nur der erste Track. Alle weiteren Tracks können das Segment erreichen.

Abbildung 3: Visualisierung der implementierten Regeln der Caesar-Gasse, der Beschränkung von Segmenten durch Wände und Engpässen.

4 Fazit zum Modul SA4E

Ich fand es sehr spannend im Rahmen des Moduls, insbesondere in den drei Übungen, Einblicke in verschiedene Frameworks und Technologien zu erhalten. Viele der Technologien, die wir uns im Rahmen des Moduls angeschaut haben, beispielsweise Apache Kafka, Apache Camel, RPC Technologien, etc., haben in der Praxis eine große Bedeutung, finden im Rahmen des Studiums allerdings kaum statt. Im Laufe des Semesters hatte ich mehrfach bei Kontakten mit ITlern aus der betrieblichen Praxis Momente, in denen Architekturen oder Technologien angesprochen wurden, die wir zuvor in der Vorlesung besprochen haben. Die Veranstaltung legt eine wichtige Grundlage für das Verständnis von Softwarearchitekturen, gibt Einblicke in einige der zentralen Herausforderungen und liefert Einblicke in wichtige Technologien. Dementsprechend bin ich sehr froh darüber, dass ich mich dazu entschieden habe, das Modul zu besuchen.