

Parallele und verteilte Systeme**Leistungsbeurteilung PVS-01**

Autor Patrick Michel
Co-Autor
Release Frühlingssemester 2024

PV-01 Transferarbeit	PV-02 Schriftliche Theorieprüfung
60%	40%

Übersicht

Personalien

Name und Vorname

Klasse

Datum

Unterschrift

Maximale Punktzahl

18 Punkte

Erreichte Punktzahl

Notenschlüssel

$$\text{Note} = \frac{\text{erreichte Punktzahl} * 5}{\text{maximale Punktzahl}} + 1$$

Note

Prüfungsform

Transferarbeit

Prüfungsdauer/Abgabetermin

14.08.2024 – 25.09.2024 23:59

Erforderliche Hilfsmittel

Open Book

weitere Vorgaben

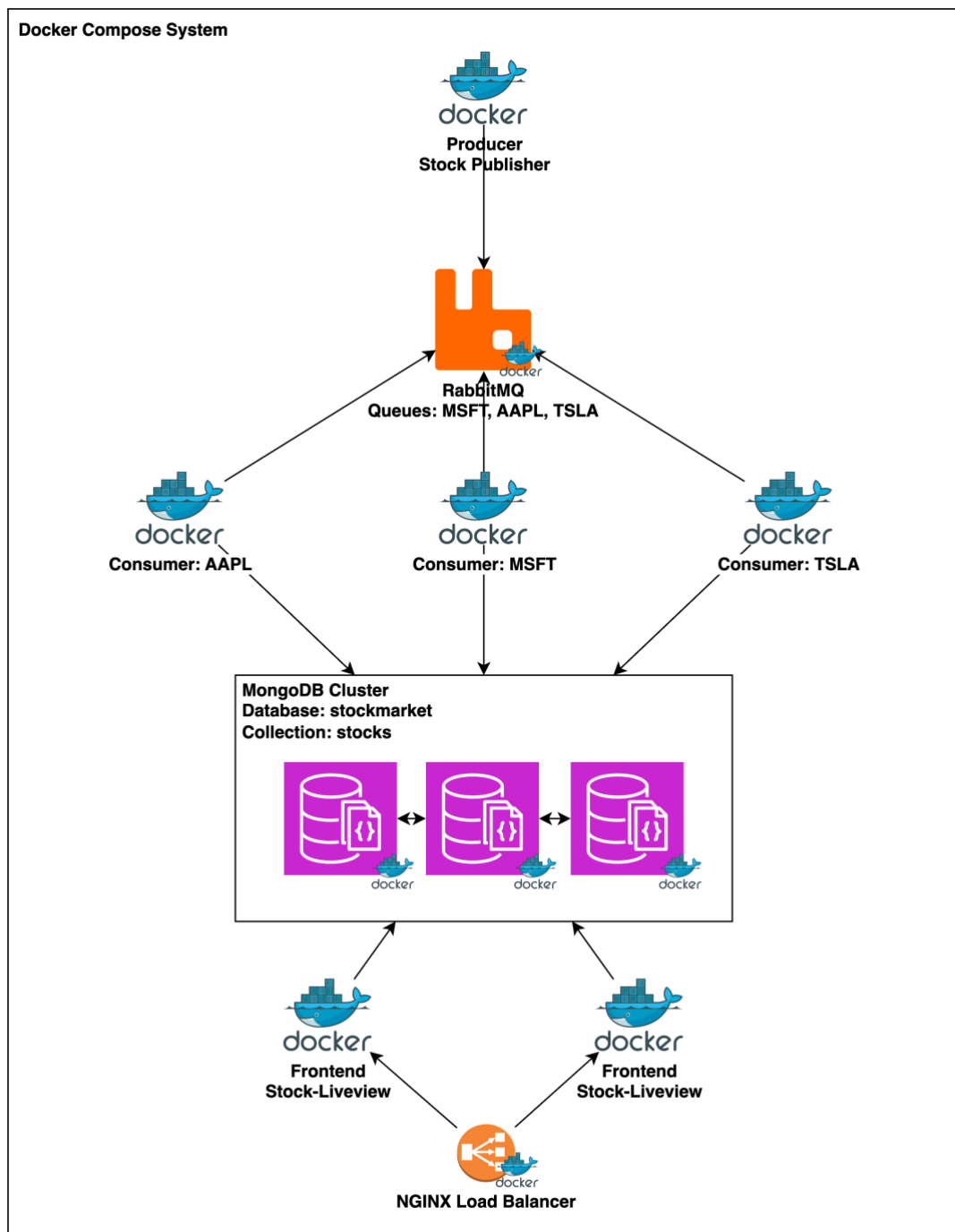
Keine

Eine Verletzung der Prüfungsbestimmungen führt zum sofortigen Abbruch der Prüfung und zur Note 1.0

Hilfsmittel: Open Book. Alle individuellen Mittel wie Computer, Bücher, eigene Notizen sind erlaubt.

Ausgangslage und Auftrag

Entwickeln Sie ein verteiltes Finanzsystem welches fiktive Finanzdaten verarbeitet und diese entsprechend abspeichert. Der Producer welcher synthetische Finanzdaten erzeugt ist als Source Code vorhanden und muss in einen Docker Container verpackt werden. Weiter sendet er die Daten in ein Message Broker System «RabbitMQ» in verschiedene Queues um diese dort für die weitere Verarbeitung zwischenspeichern. Die Consumer sollen nur jeweils eine bestimmte Gruppe von Finanzdaten verarbeiten und diese sollen aus einer spezifischen RabbitMQ Queue gelesen werden. Die Consumer sollen jeweils in 1000er Paketen Daten lesen und daraus den Durchschnitt berechnen und dieses Resultat danach in eine MongoDB speichern. Die MongoDB soll als Cluster Verbund aufgebaut sein (als Replicaset) sodass eine Ausfallsicherheit gewährleistet werden kann. Weiter sollen die aggregierten Finanzdaten über ein «Frontend» einsehbar sein welches auch als Source Code zur Verfügung gestellt wird und in einen Docker Container verpackt werden soll. Auch das Frontend soll ausfallsicher gestaltet werden und soll daher aus mindestens 2 Instanzen bestehen mit einem Load Balancer, der die Anfragen auf die Systeme verteilt.



Komponenten

Nachfolgend werden die einzelnen Komponenten des verteilten Systems beschrieben und welche Funktion diese jeweils erfüllen.

Producer «Stock-Publisher»

- Produziert zufällige «Buy» & «Sale» Datenpakete welche Finanztransaktionen auf einem Finanzmarkt von einer bestimmten Firma widerspiegeln sollen.
- Wird als Golang Applikation zur Verfügung gestellt.
 - **Code:** <https://github.com/SwitzerChees/stock-publisher>
 - **Aufgabe:** Dieser Producer soll in ein Container Image verpackt werden, sodass er in eine Docker Compose Umgebung integriert werden kann.
 - **Aufgabe:** Weiter soll das gebaute Image auf Docker Hub veröffentlicht werden.

Rabbit MQ «Message Broker»

- Dient als Zwischenspeicher zwischen Producer und Consumer und hält die vom Producer generierten Datenpakete in Queues.
 - **Aufgabe:** Soll als Container in eine Docker Compose Umgebung integriert werden, sodass dieser vom Producer Datenpakete empfangen kann
 - **Aufgabe:** Soll auch mit dem Consumer verbunden werden, sodass die Datenpakete weiterverarbeitet werden können.

Consumer

- Konsumiert die vom Producer generierten Datenpakete aus der RabbitMQ Queue und aggregiert diese sodass daraus der Durchschnitt des Preises berechnet werden kann. Das Resultat soll danach in eine MongoDB gespeichert werden.
 - **Aufgabe:** Erstellen Sie mit einer Programmiersprache Ihrer Wahl ein kleines Programm, welches in der Lage ist Nachrichten von einer spezifischen RabbitMQ Queue zu lesen (jeweils 1000 Stück), diese zu aggregieren und das Ergebnis in eine MongoDB Collection zu speichern.
 - **Aufgabe:** Die Connection Strings für die RabbitMQ und MongoDB sowie die spezifische Queue soll über Umgebungsvariablen gesteuert werden können
 - **Aufgabe:** Verpacken Sie das Programm in ein Container Image, sodass er in eine Docker Compose Umgebung integriert werden kann. Das Image soll auf Docker Hub veröffentlicht werden.

MongoDB Cluster


- Dient als Ausfallsicheres Speichersystem um die Resultate des Consumers redundant zu speichern.
 - **Aufgabe:** Soll als Container in eine Docker Compose Umgebung integriert werden, sodass dieser vom Consumer die Resultate empfangen kann.
 - **Aufgabe:** Soll auch mit dem Frontend verbunden werden, sodass die Resultate angezeigt werden können.

Frontend «Stock-Liveview»

- Zeigt die Resultate, welche in die MongoDB gespeichert werden als Livestream an.
- Wird als NodeJS Applikation zur Verfügung gestellt.
 - **Code:** <https://github.com/SwitzerChees/stock-liveview>
 - **Aufgabe:** Dieses Frontend soll in ein Container Image verpackt werden, sodass er in eine Docker Compose Umgebung integriert werden kann.
 - **Aufgabe:** Weiter soll das gebaute Image auf Docker Hub veröffentlicht werden.

NGINX «Load Balancer»

- Dient als Lastverteilung und als Failover für das Frontend.
 - **Aufgabe:** Soll als Container in eine Docker Compose Umgebung integriert werden, sodass dieser mit den Frontends kommunizieren kann.
 - **Aufgabe:** Soll die Anfragen auf mindestens 2 Frontends verteilen und bei einem Ausfall ein sauberes Failover machen.

 Falls von dem Studierenden gewollt können auch andere Produkte eingesetzt werden, das muss aber zuerst mit der Lehrperson besprochen werden.

Datenstruktur

Nachfolgend werden die Datenstrukturen beschrieben, an die man sich halten sollte, damit der «Producer» und auch das «Frontend» wie erwartet funktionieren.

Rabbit MQ «Message Broker»

Die Queues «AAPL», «MSFT» und «TSLA» werden automatisch erzeugt, sobald der «stock-publisher» gestartet wird. Das Format der einzelnen Nachrichten ist wie folgt (als JSON Format):

- **company:** Der Name der Firma.
- **eventType:** Die Art des Events buy/sell.
- **price:** Der zufällig generierte Preis.

Message 1

The server reported 4668 messages remaining.

Exchange	(AMQP default)
Routing Key	AAPL
Redelivered	0
Properties	content_type: application/json
Payload	{ "company": "AAPL", "eventType": "sell", "price": 64.52611706553348 }

63 bytes
Encoding: string

Folgende Ansicht sollte in der RabbitMQ zu sehen sein:

<http://localhost:15672/>

RabbitMQ™

RabbitMQ 3.13.2

Erlang 26.2.5

Overview

Connections

Channels

Exchanges

Queues and Streams

Admin

Queues

▼ All queues (3)

Pagination

Page 1 of 1 - Filter: ☐ Regexp ?

Overview

				Messages			Message rates				+/-
Virtual host	Name	Type	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack	
/	AAPL	classic		<div>running</div>	24,995	0	24,995	1,000/s			
/	MSFT	classic		<div>running</div>	24,999	0	24,999	1,000/s			
/	TSLA	classic		<div>running</div>	24,998	0	24,998	1,000/s			

MongoDB Cluster

- **Datenbank:** stockmarket
- **Collection:** stocks
- **Dokumentenschema:**
 - **_id:** Standard Id Format von MongoDB verwenden
 - **company:** Bezeichnung der Firma als String
 - **avgPrice:** Aktuell berechneter Preis als «Double» im Gleitkommazahlen zu erlauben.

Document	_id	company	avgPrice
1	ObjectId('663e42a9e5fb13bb45d9816c')	TSLA	123.45
2	ObjectId('663e436be5fb13bb45d9816d')	MSFT	245.6
3	ObjectId('663e437ee5fb13bb45d9816e')	AAPL	345.1

Artefakte und Bewertungsraster

Nachfolgend finden Sie eine Auflistung der Kriterien, welche erfüllt sein müssen und die benotet werden.

⚠ Denken Sie daran jedes Kriterium muss von der Lehrperson nachvollzogen werden können daher sollte in der Abgabe alles, was Sie erarbeitet haben, vorhanden sein. Siehe dazu «Geforderte Abgaben».

- **(0-2 Punkte) Producer «Stock-Publisher»**
 - (0-1 Punkt) Ist in einem Docker Compose als Container Image integriert und funktioniert wie erwartet.
 - (0-1 Punkt) Ist auf Docker Hub als Image veröffentlicht.
- **(0-2 Punkte) RabbitMQ «Message Broker»**
 - (0-1 Punkt) Ist in einem Docker Compose als Container Image integriert und funktioniert wie erwartet.
 - (0-1 Punkt) Hält mehrere Queues mit den Datenpaketen des Producers und löscht diese, nachdem sie vom Consumer gelesen wurden aus der Queue.
- **(0-4 Punkte) Consumer**
 - (0-1 Punkt) Ist in einem Docker Compose als Container Image integriert. Jeweils 1 Container pro Firma (3x).
 - (0-1 Punkt) Konsumiert jeweils 1000 Datenpakete aus RabbitMQ von einer konfigurierbaren Queue.
 - (0-1 Punkt) Aggregiert die Datenpakete und speichert diese in eine MongoDB.
 - (0-1 Punkt) Ist auf Docker Hub als Image veröffentlicht.
- **(0-4 Punkte) MongoDB Cluster**
 - (0-1 Punkt) Ist in einem Docker Compose als Container Image und in einem Replicaset Setup integriert. Jeweils 1 Container pro Node (3x).
 - (0-1 Punkt) Ist in der Lage mit dem Consumer sowie dem Frontend zu kommunizieren, um die notwendigen Daten auszutauschen.
 - (0-1 Punkt) Das System läuft weiter, auch wenn 1 der Nodes heruntergefahren wird.
 - (0-1 Punkt) Die Daten werden mithilfe von Volume persistiert und bleiben nach einem Neustart vorhanden.
- **(0-3 Punkte) Frontend «Stock-Liveview»**
 - (0-1 Punkt) Ist in einem Docker Compose als Container Image integriert und funktioniert wie erwartet.
 - (0-1 Punkt) Das System läuft weiter, auch wenn 1 der Frontends heruntergefahren wird.
 - (0-1 Punkt) Ist auf Docker Hub als Image veröffentlicht.
- **(0-1 Punkt) NGINX «Load Balancer»**
 - (0-1 Punkt) Ist in einem Docker Compose als Container Image integriert und funktioniert wie erwartet. Das heisst er verteilt Anfragen sauber auf die Frontends und macht bei einem Ausfall ein automatisches Failover.
- **(0-2 Punkte) Versionsverwaltung mit «git».**
 - (0-1 Punkt) «git» Repository auf Github vorhanden & Commit Kommentare sind sinnvoll sowie verständlich.
 - (0-1 Punkt) Commits werden regelmässig über den Zeitraum der Umsetzung erstellt.

Geforderte Abgaben

Folgende Abgaben sind nach Erfüllung der Aufgaben zu tätigen:

- Link zu Github Repository