الجمهورية العربية السورية
المعهد العالي للعلوم التطبيقية والتكنولوجيا
قسم النظم المعلوماتية
العام الدراسي 2024/2025

مشروع تخرج

أعد لنيل درجة الإجازة في هندسة الشبكات ونظم التشغيل

تحليل تقييمات الأفلام واكتشاف الاتجاهات الجديدة باستخدام نظام المعطيات الكبيرة Hadoop

تقديم

امير احمد يوسف

إشراف

د. حسين خيو

30/7/2025

الخلاصة

لمواجهة تحديات أنظمة توصية التقليدية المتمثلة في تأخر الاستجابة وقابلية التوسع المحدودة، يهدف هذا المشروع إلى تصميم وتنفيذ بنية نظام توصية هجين ومتقدم للأفلام. إذ يدمج النظام بين الدقة التحليلية للمعالجة الدفعية والاستجابة الفورية للمعالجة في الزمن الحقيقي. تم بناء طبقة دفعية باستخدام Apache Spark لتدريب نموذج ترشيح تعاوني قائم على تحليل العوامل (ALS) بشكل دوري، بالإضافة إلى إجراء تحليلات استكشافية معمقة للبيانات. بالتوازي، تم بناء طبقة معالجة لحظية باستخدام Apache Flink و Apache Kafka، تقوم باستهلاك تفاعلات المستخدم الآنية وتحديث التوصيات. تم تصميم البنية الكلية وفق نمط الخدمات المصغرة، مع اتباع كل خدمة لمبادئ البنية السداسية (Hexagonal Architecture) لضمان قابلية الصيانة والاختبار. تم نشر النظام في بيئة حاويات باستخدام Docker و Kubernetes و DataProc على منصة Google Cloud Platform، وأظهرت نتائج الاختبارات كفاءة طبقة المعالجة الدفعية وقدرتها على التوسع، كما سلطت الضوء على متطلبات البنية التحتية اللازمة لتحقيق زمن استجابة منخفض في الطبقة اللحظية.

Abstract

To address the challenges of traditional recommendation systems, namely response latency and limited scalability, this project aims to design and implement advanced hybrid movie recommendation system architecture. The system integrates the analytical accuracy of batch processing with the immediate responsiveness of real-time processing. A batch layer was built using Apache Spark to periodically train a collaborative filtering model based on matrix factorization (ALS) and perform in-depth exploratory data analysis. In parallel, a real-time layer was constructed using Apache Flink and Apache Kafka to consume user interactions as they occur and update recommendations instantly. The overall architecture was designed following the microservices pattern, with each service adhering to the principles of Hexagonal Architecture to ensure maintainability and testability. The system was deployed in a containerized environment using Docker and Kubernetes on the Google Cloud Platform. Test results

demonstrated the efficiency and scalability of the batch processing layer and highlighted the infrastructure requirements needed to achieve low latency in the real-time layer.

المحتويات

قائمة الصور

قائمة الجداول

الفصل الأول

التعريف بالمشروع

يتضمن هذا الفصل التعريف بالمشروع ومتطلباته.

1.1- مقدمة

في عصر تدفق المعلومات والبيانات الضخمة، يواجه مستخدمو منصات بث المحتوى الرقمي، وخاصة منصات الأفلام والفيديو، تحدياً متزايداً يُعرف بـ«فيض المعلومات». فالكمُّ الهائل من الخيارات المتاحة يجعل من الصعب على المستخدمين اكتشاف المحتوى الذي يتوافق بدقة مع أذواقهم وتفضيلاتهم الشخصية. استجابةً لهذا التحدي، برزت أنظمة التوصية كأدوات حيوية وضرورية لتحسين تجربة المستخدم وزيادة مستوى تفاعله وولائه للمنصات الرقمية، من خلال تقديم اقتراحات مخصصة وذات صلة عالية. [1]

تعتمد العديد من أنظمة التوصية التقليدية على نماذج المعالجة الدفعية(Batch Processing) ، حيث يتم تحليل البيانات المخزنة بشكل دوري لتحديث التوصيات. وعلى الرغم من قدرة هذا النهج على بناء نماذج دقيقة، فإنه يؤدي إلى تأخير ملحوظ في الاستجابة للتغيرات اللحظية في اهتمامات المستخدم أو تفاعله مع المحتوى الجديد. بالإضافة إلى ذلك، تواجه هذه الأنظمة تحديات جوهرية تتعلق بقابلية التوسع (Scalability) اللازمة للتعامل مع ملايين المستخدمين ومليارات التفاعلات والتقييمات؛ فضلاً عن «مشكلة البداية الباردة» (Cold Start Problem) التي تحد من قدرتها على تقديم توصيات فعّالة لتغير توجهات المستخدمين ونواياهم.[2]

يهدف هذا المشروع إلى تقديم حل مبتكر لهذه التحديات من خلال تصميم وتطوير نظام توصية هجين يدمج بين قوة المعالجة الدفعية لتحليل البيانات وبناء نماذج توصية دقيقة، وبين مرونة المعالجة في الزمن الحقيقي للاستجابة الفورية لتفاعلات المستخدم وتحديث التوصيات بشكل آني.

2.1- هدف المشروع

يسعى النظام المقترح إلى تحسين تجربة مستخدمي منصات الأفلام عبر تقديم توصيات شخصية ذات دقّة عالية واستجابة آنية لتفاعلات المستخدم. يتحقق ذلك من خلال بنية تحتية تعتمد على تقنيات البيانات الضخمة، حيث تُستخدم المعالجة الدفعية لبناء وتحديث نماذج التوصية الأساسية بشكل دوري، ثم تُفعّل المعالجة في الزمن الحقيقي (Real-time Processing) لتكييف هذه النماذج وتحديثها فورياً استناداً إلى سلوك المستخدم اللحظي، مما يجعل النظام قابلاً للتوسع وفعالاً في مواكبة التغيرات اللحظية لتفضيلات المستخدمين وحجم البيانات الضخم والمتغير باستمرار.

3.1- المتطلبات الوظيفية

يجب أن يقدم النظام ما يلي:

1- **استيعاب وتخزين البيانات :**يجب أن يكون لدى النظام القدرة على استيعاب كميات ضخمة من بيانات تفاعلات المستخدمين مع الأفلام، مثل التقييمات والمشاهدات، إلى جانب البيانات الوصفية الخاصة بالأفلام والمستخدمين، وتخزينها بشكل منظم يتيح الوصول إليها ومعالجتها لاحقاً.

2- **المعالجة المسبقة للبيانات :**لا بد من توفير آلية فعّالة لتنظيف وتحضير البيانات المخزنة، بما يشمل إزالة القيم الشاذة أو الناقصة وتوحيد الصيغ، لجعلها مناسبة لعملية تدريب نموذج التوصية.

3- **تحليل البيانات:** يجب أن يمتلك النظام القدرة على تحليل البيانات المخزنة واستخراج الإحصائيات المختلفة والمتعلقة بنشاط المستخدمين والأفلام الأكثر شهرة وشعبية وأنواعها، وذلك لتمكين محللي البيانات من اتخاذ قرارات تتعلق بالتوجهات والتوقعات المستقبلية فيما يخص مجال الأفلام.

4- **التدريب لنموذج التوصية :**ينبغي أن يمتلك النظام القدرة على تدريب نموذج ترشيح تعاوني، باستخدام البيانات المخزنة مسبقاً.

5- **تخزين مخرجات النموذج :**من المهم تخزين العوامل الكامنة (Latent Factors) للمستخدمين والأفلام الناتجة عن عملية التدريب في مخزن بيانات سريع الوصول، لتُستخدم لاحقاً في توليد التوصيات سواء الدفعية أو اللحظية.

6- **التقاط الأحداث اللحظية :**يجب أن يتمكن النظام من استقبال ومعالجة تدفقات بيانات تفاعلات المستخدم الجديدة (مثل تقييم فلم) في اللحظة التي تحدث فيها، مما يتيح تحديث التوصيات بسرعة.

7- **توليد التوصيات الأساسية (الدفعية) :**يجب أن يكون بالإمكان توليد قائمة توصيات مخصصة لكل مستخدم بناءً على نتائج النموذج المدرب دورياً، وهي تمثل الأساس العام للتوصيات.

8- **توليد/تحديث التوصيات اللحظية/ذات حالة :**يجب أن يكون لدى النظام القدرة على انشاء التوصيات بشكل لحظي بناءً على تفاعلات المستخدم **الجديدة**، وذلك بالاستفادة من النموذج الأساسي المخزن لتعزيز دقة التوصيات في الزمن الحقيقي.

4.1- المتطلبات غير الوظيفية

1- **قابلية التوسع (Scalability)**: يجب أن يكون النظام قادراً على مواكبة الزيادة المستمرة في حجم البيانات وعدد المستخدمين ومعدل التفاعلات اللحظية دون تدهور في الأداء.

2- **الموثوقية (Reliability)**: يجب أن يعمل النظام بشكل مستقر وأن يتحمل الأعطال الجزئية في بعض مكوناته دون التأثير على توفر الخدمة أو سلامة البيانات.

3- **الدقة (Accuracy)**: ينبغي أن يقدّم نموذج التوصية الدفعية مستوى عالي من الدقة يمكن قياسه ومقارنته باستخدام مؤشرات أداء مثل RMSE.

4- **قابلية الصيانة (Maintainability)**: يجب أن يكون التصميم المعماري واضحاً والكود المصدري منظّماً، بما يسهل عمليات التصحيح والتحديث وإضافة ميزات جديدة في المستقبل.

تقديم واجهة للحصول على التوصيات: وأخيراً، يجب توفير واجهة برمجة تطبيقات (API) تمكّن الأنظمة الأخرى أو واجهات المستخدم من طلب التوصيات واسترجاع أحدث النتائج الخاصة بأي مستخدم بسهولة وفعالية.

الفصل الثاني

الدراسة التحليلية

يوضح هذا الفصل عمليّة تحليل النظام ودراسة متطلباته.

1.2- تحديد الفاعلين (Actors)

- **المستخدم النهائي (End User)**

المستفيد الأساسي من النظام، والذي يتلقى توصيات الأفلام ويتفاعل معها (التقييمات، المشاهدات، النقرات) كمدخلات للنظام.

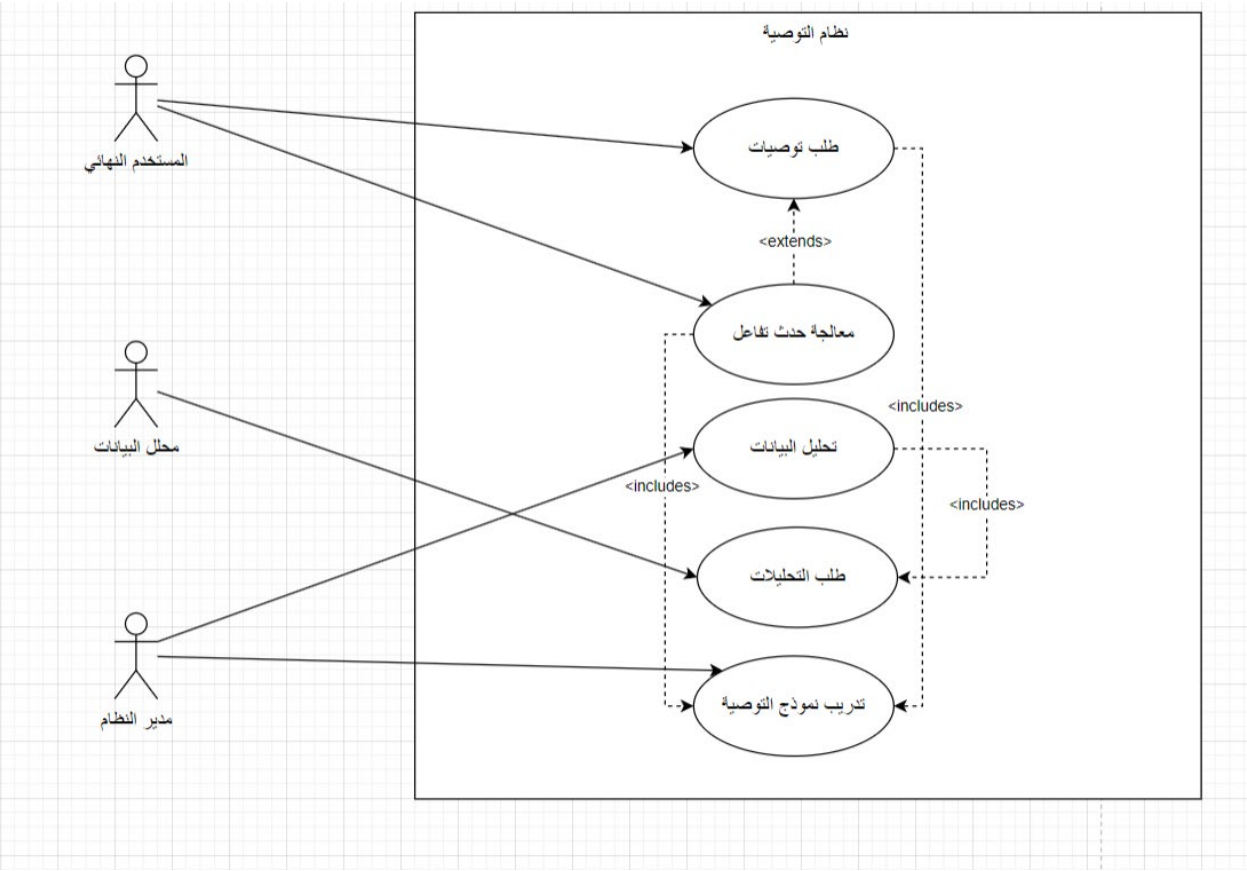
- **مدير النظام / المشغّل (System Administrator / Operator)**

الشخص القائم على إدارة البنية التحتية، جدولة عمليات التدريب وإجراء التحليلات، ومراقبة أداء مكوّنات النظام.

- **محلل البيانات (Data Analyst)**

الشخص القائم على قراءة تحليلات البيانات واتخاذ الإجراءات المناسبة تباعاً.

2.2- مخطط حالات الاستخدام (Use Case Diagrams)



صورة **1** مخطط حالات الاستخدام

2.2- السرد النصي لحالات الاستخدام

1.2.2- تدريب نموذج التوصية الدفعية

اسم الحالة: تدريب نموذج التوصية الدفعية	
الوصف Description	يقوم النظام بشكل دوري بتحليل البيانات المخزنة لتفاعلات المستخدمين والأفلام لتدريب وتحديث نموذج الترشيح التعاوني (ALS) الأساسي.
الفاعلين Actors	مدير النظام (يراقب العملية)، النظام (ينفذ العملية).

الشروط السابقة Precondition	– توفر بيانات كافية في نظام التخزين الموزع. – توفر موارد حوسبة كافية في عنقود المعالجة الدفعية. – وجود جدول زمني محدد لتشغيل المهمة.
الشروط اللاحقة Postcondition	– تم تدريب نموذج ALS محدث. – تم حساب مقاييس تقييم النموذج (مثل RMSE). – تم تصدير عوامل المستخدمين والأفلام الناتجة إلى مخزن البيانات السريع.

جدول 1: تدريب نموذج التوصية الدفعية

سير الأحداث

السيناريو الأساسي الناجح

مدير النظام	النظام (طبقة المعالجة الدفعية)
1. يقوم بتشغيل مهمة تدريب النموذج بناءً على جدول زمني أو بشكل مباشر.	
	2. تبدأ مهمة المعالجة.
	3. تقرأ البيانات المُجهزة من نظام التخزين الموزع.
	4. تقوم بتدريب نموذج ALS باستخدام البيانات المقروءة والمعاملات المحددة.
	5. تستخرج عوامل المستخدمين (User Factors) وعوامل الأفلام (Item Factors) من النموذج المدرب.
	6. يقوم النموذج المنشأ بتوليد توصيات لكل المستخدمين الحاليين وحفظها في قاعدة معطيات.
	7. تتصل بمخزن البيانات السريع.

	8. تكتب/تُحدّث عوامل المستخدمين والأفلام في مخزن البيانات السريع.
	9. تُنهي المهمة بنجاح وتسجل النتائج (مثل RMSE، وقت التنفيذ).

جدول 2: السيناريو الناجح لحالة استخدام تدريب نموذج التوصية الدفعية.

المسارات البديلة

لا يوجد.

مسارات الأخطاء

E1: في المرحلة 3، فشل قراءة البيانات من نظام التخزين (مشكلة اتصال، بيانات تالفة).

يقوم النظام بتسجيل الخطأ وإنهاء المهمة بفشل.

E2: في المرحلة 4، حدوث خطأ أثناء تدريب النموذج (نقص موارد، خطأ في الخوارزمية).

يقوم النظام بتسجيل الخطأ وإنهاء المهمة بفشل.

E3: في المرحلة 7 أو 8، فشل الاتصال أو الكتابة إلى مخزن البيانات السريع.

يقوم النظام بتسجيل الخطأ ومحاولة إعادة الاتصال/الكتابة لعدد محدد من المرات، ثم إنهاء المهمة بفشل إذا استمر الخطأ.

2.2.2- معالجة حدث تفاعل لحظي

اسم الحالة: معالجة حدث تفاعل لحظي	
الوصف Description	يقوم النظام باستقبال ومعالجة حدث تفاعل مستخدم جديد فور وصوله لتحديث التوصيات المحتملة لهذا المستخدم أو تكييف حالته.
الفاعلين Actors	المستخدم، النظام.
الشروط السابقة Precondition	– طبقة المعالجة اللحظية تعمل وتستمع للأحداث.

	– مخزن البيانات السريع يحتوي على عوامل نموذج محدثة (من الطبقة الدفعية).
الشروط اللاحقة Postcondition	– تم استهلاك الحدث ومعالجته بواسطة طبقة المعالجة اللحظية. – تم حساب توصيات محدثة أو تعديل حالة مرتبطة بالمستخدم بناءً على الحدث ضمن الطبقة اللحظية. – تم إرسال النتائج إلى الوجهة المحددة.

جدول 3: معالجة حدث تفاعل مستخدم لحظي.

سير الأحداث

السيناريو الأساسي الناجح

الفاعل (المستخدم)	النظام
1. يرسل حدث تفاعل جديد (مثل تقييم فيلم).	
	2. تستهلك طبقة المعالجة اللحظية الحدث.
	3. تقوم بتحليل (Parse) بيانات الحدث واستخراج المعلومات الهامة (userId, movieId, rating/eventType).
	4. تتصل بمخزن البيانات السريع.
	5. تجلب عامل المستخدم (User Factor) للمستخدم المحدد (userId).

	6. تجلب عامل الفيلم (Item Factor) للفيلم المحدد (movieId) أو عوامل الأفلام المرشحة للتوصية.
	7. تقوم بحساب درجات التوصية (Recommendation Scores) المحدثة باستخدام العوامل التي تم جلبها وبيانات الحدث. تؤكد هذه الخطوة على دور الطبقة اللحظية في استخدام مخرجات الطبقة الدفعية (العوامل) وتطبيق منطق التقييم الفوري.
	8. ترسل النتائج (مثل قائمة التوصيات المحدثة أو درجاتها) إلى مكون الإخراج.
	9. يقوم مكون الإخراج بإرسال النتائج إلى الوجهة المحددة.

جدول 4: السيناريو الناجح لحالة استخدام معالجة حدث تفاعل لحظي.

المسارات البديلة

- **A1: في المرحلة 7، قد لا يتم حساب توصيات جديدة لكل حدث، بل قد يتم فقط تحديث حالة وسيطة للمستخدم (مثل قائمة الأفلام التي تفاعل معها مؤخراً) ضمن الطبقة اللحظية لاستخدامها لاحقاً.**
- **A2: في المرحلة 5 أو 6، قد لا يتم العثور على عامل للمستخدم أو الفيلم في Redis. قد يطبق النظام منطقاً احتياطياً (Fallback logic) ضمن الطبقة اللحظية.**

مسارات الأخطاء

E1: في المرحلة 3، فشل تحليل بيانات الحدث.

تسجل الطبقة اللحظية الخطأ وقد تتجاهل الحدث أو ترسله لقائمة أخطاء.

E2: في المرحلة 4 أو 5 أو 6، فشل الاتصال أو القراءة من مخزن البيانات السريع.

تسجل الطبقة اللحظية الخطأ وتحاول إعادة المحاولة أو تتجاهل الحدث.

E3: في المرحلة 7، حدوث خطأ أثناء حساب التوصيات ضمن الطبقة اللحظية.

تسجل الطبقة اللحظية الخطأ.

E4: في المرحلة 9، فشل إرسال النتائج من الطبقة اللحظية.

تسجل الطبقة اللحظية الخطأ وتحاول إعادة الإرسال.

3.2.2- طلب توصيات لمستخدم

اسم الحالة: طلب توصيات لمستخدم	
الوصف Description	يقوم المستخدم بطلب قائمة بأحدث توصيات الأفلام لمستخدم معين من نظام التوصية.
الفاعلين Actors	المستخدم، النظام
الشروط السابقة Precondition	– وجود واجهة API معرفة لاستقبال طلبات التوصية. – توفر توصيات المستخدم في مخزن البيانات.
الشروط اللاحقة Postcondition	تم إرجاع قائمة بأفضل توصيات أفلام للمستخدم المحدد إلى التطبيق المستهلك.

جدول 5: حالة استخدام طلب توصيات لمستخدم.

سير الأحداث

السيناريو الأساسي الناجح

الفاعل (التطبيق المستهلِك)	النظام (واجهة API + الطبقة اللحظية/الدفعية)
1. يرسل طلباً إلى واجهة API التوصية، محدداً معرف المستخدم (userId).	
	2. تستقبل واجهة API الطلب وتتحقق من صحته.
	3. تتصل بمخزن البيانات لجلب توصيات المستخدم المحسوبة مسبقاً للمستخدم المحدد.
	4. (إذا تم العثور على عامل المستخدم) تجلب التوصيات المقترحة للمستخدم.
	5. تعيد قائمة بالتقييمات إلى التطبيق المستهلِك عبر استجابة API.

جدول 6: السيناريو الناجح لحالة استخدام طلب توصيات لمستخدم.

المسارات البديلة

A1: في المرحلة 3، إذا لم يتم العثور على معرف المستخدم (حالة بداية باردة) ، قد يلجأ النظام إلى:

إرجاع توصيات عامة (الأكثر شيوعاً).

مسارات الأخطاء

E مسارات الأخطاء (Error Paths):

E1: في المرحلة 2، فشل التحقق من صحة الطلب.

تعيد واجهة API استجابة خطأ مناسبة.

E2: **في المرحلة 3 أو 4، فشل الاتصال أو القراءة من مخزن البيانات**.

يسجل النظام الخطأ وقد يحاول تطبيق منطق احتياطي (Fallback) أو يعيد استجابة خطأ.

الفصل الثالث

تصميم النظام

يعرض هذا الفصل القرارات التصميمية التي بني من خلالها النظام.

1.3- مقدمة

لتحقيق الأهداف والمتطلبات التي تم تحديدها في الفصول السابقة، وخاصة متطلبات قابلية التوسع، الصيانة، والنشر المستقل، تم اعتماد بنية الخدمات المصغرة (Microservices Architecture) كأساس لتصميم النظام. يتكون النظام من مجموعة من الخدمات المستقلة التي تعمل مع بعضها بشكل متكامل لخدمة الأهداف الوظيفية المختلفة. كل خدمة مسؤولة عن نطاق محدد من العمل (Bounded Context)، ويمكن تطويرها، نشرها، وتوسيعها بشكل مستقل عن الخدمات الأخرى. [7]

تقدم هذه البنية مجموعة من الفوائد الجوهرية لمشروعنا:

- **التوسع المستقل (Independent Scaling)**: يمكن توسيع كل خدمة مصغرة (مثل خدمة المعالجة اللحظية أو خدمة تحليل البيانات) بشكل مستقل بناءً على الحمل الذي تتعرض له. تسمح هذه المرونة باستخدام الموارد بكفاءة، حيث يتم تخصيص الموارد فقط للأجزاء التي تحتاج إليها. [7]
- **المرونة في استخدام التقنيات (Technological Flexibility)**: يمكن استخدام لغات البرمجة وأطر العمل وقواعد البيانات الأنسب لكل خدمة بناءً على متطلباتها المحددة. على سبيل المثال، استخدام Apache Spark للمهام الدفعية الثقيلة، وApache Flink للمهام اللحظية التي تتطلب زمن استجابة منخفض، وMongoDB لتخزين نتائج التحليل شبه المهيكلة. [7]
- **النشر المستقل (Independent Deployment)**: يمكن نشر تحديثات أو إصلاحات لأي خدمة دون الحاجة إلى إعادة نشر النظام بأكمله. يدعم هذا بشكل مباشر ممارسات التكامل المستمر والتسليم المستمر (CI/CD)، مما يسرع من وتيرة التطوير ويقلل من المخاطر المرتبطة بعمليات النشر الكبيرة. [7]

بالإضافة إلى ذلك، ستتبع كل خدمة مصغرة داخلياً مبادئ البنية السداسية (Hexagonal Architecture)، أو نمط المنافذ والمحولات (Ports and Adapters)، لعزل منطق العمل الأساسي (النواة) عن تفاصيل البنية التحتية الخارجية (مثل قواعد البيانات، أنظمة الرسائل، أو أطر العمل المحددة)، مما يعزز قابلية الاختبار والصيانة بشكل كبير داخل كل خدمة.

قبل الخوض في تفاصيل كل خدمة، تجدر الإشارة إلى وجود مكتبة مشتركة (shared-kernel-library) تحتوي على نماذج النطاق الأساسية المشتركة بين الخدمات، والتي تم تصميمها لتكون صغيرة ومستقرة لتجنب الاقتران غير الضروري. نعرض في الفقرات التالية شرحاً لتصميم كل خدمة من الخدمات الرئيسية التي تستفيد من هذه النواة المشتركة.

2.3- خدمة المعالجة الدفعية

المسؤولية الأساسية: هذه الخدمة الموحدة مسؤولة عن جميع عمليات المعالجة الدفعية التي تتم بشكل دوري. تم دمج مسؤوليات تدريب نموذج التوصية وتحليل البيانات في هذه الخدمة للاستفادة من خطوات تحميل ومعالجة البيانات المشتركة وتسهيل الإدارة التشغيلية، حيث إن كلا المهمتين تعتمدان على نفس مجموعة البيانات الأولية ونفس تقنية المعالجة (Apache Spark). وتشمل مسؤولياتها:

- تدريب نموذج التوصية (ALS) وتحديث مخزن عوامل المستخدمين والأفلام في Redis.
- إجراء تحليلات استكشافية وإحصائية على بيانات MovieLens وتخزين النتائج المجمعة في MongoDB.

تصميم البنية الداخلية (سداسية):

- النواة (Domain & Application Core): يتم عزل منطق العمل الجوهري للخدمة عن تفاصيل تقنية Spark أو كيفية تخزين البيانات:
  - النطاق (Domain): تحتوي هذه الطبقة على نماذج البيانات والمفاهيم الأساسية المستقلة، مثل تعريف "التقييم المعالج"، "عامل المستخدم"، "عامل الفيلم"، بالإضافة إلى نماذج تمثل مخرجات التحليل مثل "شعبية الأنواع" أو "اتجاهات التقييم".
  - التطبيق (Application): تحتوي هذه الطبقة على منطق حالات الاستخدام (Use Cases) وتعرف المنافذ (Ports) كواجهات برمجية. المنافذ تحدد العقود التي تتفاعل بها نواة التطبيق مع العالم الخارجي، مثل منفذ لتحميل بيانات MovieLens، منفذ لحفظ عوامل النموذج، ومنفذ لحفظ نتائج التحليل. كما تعرف منافذ الإدخال التي تبدأ عمليات التدريب والتحليل.
- المحولات (Adapters): هي التطبيقات الملموسة للمنافذ التي تتعامل مع التقنيات الخارجية:

- محول التشغيل (Driving Adapter): يتمثل في مهمة Spark الرئيسية. هذه المهمة هي التي تبدأ العملية (بناءً على جدولة أو استدعاء يدوي)، وتقوم بتنسيق العمل من خلال استدعاء حالات الاستخدام المعرفة في طبقة التطبيق. هي مسؤولة عن تهيئة بيئة Spark وحقن المحولات اللازمة في خدمات التطبيق.
- محولات الإخراج (Driven Adapters):
  - محول الوصول للبيانات: ينفذ منفذ تحميل البيانات باستخدام Spark SQL وDataFrame API لقراءة البيانات من HDFS.
  - محول تخزين العوامل: ينفذ منفذ حفظ العوامل عن طريق كتابة مخرجات نموذج ALS إلى Redis.
  - محول تخزين التقييمات: ينفذ منفذ حفظ التقييمات بالكتابة والتخزين على MongoDB.
  - محول تخزين التحليلات: ينفذ منفذ حفظ نتائج التحليل عن طريق كتابة مخرجات الاستعلامات التحليلية إلى MongoDB.

التقنية الأساسية: Apache Spark.

3.3- خدمة المعالجة اللحظية

المسؤولية الأساسية: استهلاك تدفقات أحداث تفاعل المستخدمين في الزمن الحقيقي، واستخدام عوامل النموذج لإجراء **بحث عن المتجهات**، وتوليد توصيات لحظية مخصصة.

تصميم البنية الداخلية (سداسية):

- النواة (Domain & Application Core):
  - النطاق (Domain): تحتوي على نماذج المفاهيم الأساسية للمعالجة اللحظية مثل "حدث التفاعل"، "الفيلم المرشح" (الناتج من بحث المتجهات)، و "التوصية الشخصية" (النتيجة النهائية).
  - التطبيق (Application):
    - منافذ الإدخال: تُعرف واجهة تحدد حالة استخدام معالجة حدث وارد.
    - منافذ الإخراج: تُعرف واجهات مجردة لجلب عوامل النموذج من مخزن خارجي، ولإجراء بحث متجهي عن العناصر المتشابهة، ولنشر التوصيات الناتجة.

- خدمات التطبيق: تحتوي على منطق العمل الذي ينسق الخطوات: جلب العوامل، طلب المرشحين عبر بحث المتجهات، حساب الدرجات الشخصية، ترشيح النتائج، وإصدار التوصية النهائية عبر منفذ الإخراج.

- المحولات (Adapters):

  - محول التشغيل (Driving Adapter): يتمثل في مهمة Apache Flink. تعمل هذه المهمة كمحول إدخال يستهلك الأحداث من نظام الرسائل (محول ضمني)، وينسق استدعاء حالة الاستخدام المعرفة في طبقة التطبيق لكل حدث يتم معالجته.

  - محولات الإخراج (Driven Adapters):

    - محول جلب العوامل: ينفذ منفذ جلب العوامل بالاتصال بـ Redis.
    - محول بحث المتجهات: ينفذ منفذ بحث المتجهات بإرسال أوامر البحث (مثل KNN) إلى RediSearch.
    - محول نشر النتائج: ينفذ منفذ نشر النتائج بإرسال التوصيات النهائية إلى نظام رسائل آخر مثل Pub/Sub.

التقنية الأساسية: تم اختيار Apache Flink كتقنية أساسية لهذه الخدمة.

4.3- خدمة الاستعلام عن التقييمات

تعتبر هذه الخدمة من الخدمات **الثانوية** في النظام، إذ هي مجرد طريقة أو بوابة تقدم للمنصات لتمكينها من معرفة تقييمات المستخدمين المرتبطين بهذه المنصة والذي تم إجراء تحليل لسلوكهم عن طريق النموذج المدرب.

تم استخدام منهجية البنية سداسية الشكل (hexagonal architecture)، التي تقدم مجموعة من الخواص أهمها:

- عدم الاعتماد على التفاصيل التنجيزية الخارجية في معالجة منطق العمل (Business logic) حيث نستطيع تغيير هذه التفاصيل (مثل قاعدة المعطيات، مخدمات الرسائل، كيفية التواصل مع الخدمات الأخرى ...) دون الحاجة لتعديل باقي أجزاء النظام.
- تكون مركبات النظام غير مرتبطة مع بعضها البعض بقوة (Loosely coupled) مما يقدم إمكانية التعديل والصيانة بسهولة.

5.3- خدمة الاستعلام عن احصائيات البيانات

هذه الخدمة حالها كحال خدمة الاستعلام عن التقييمات، موجودة لتمكين أصحاب المصلحة وعلماء البيانات من الاطلاع على الإحصائيات التي تخص البيانات المعالجة من معرفة توجهات المستخدمين وتحديد الأفلام الأكثر شهرة وخصائصها وذلك بغية اتخاذ قرارات مستقبلية تخص السياسات التي يجب اتباعها في المستقبل.

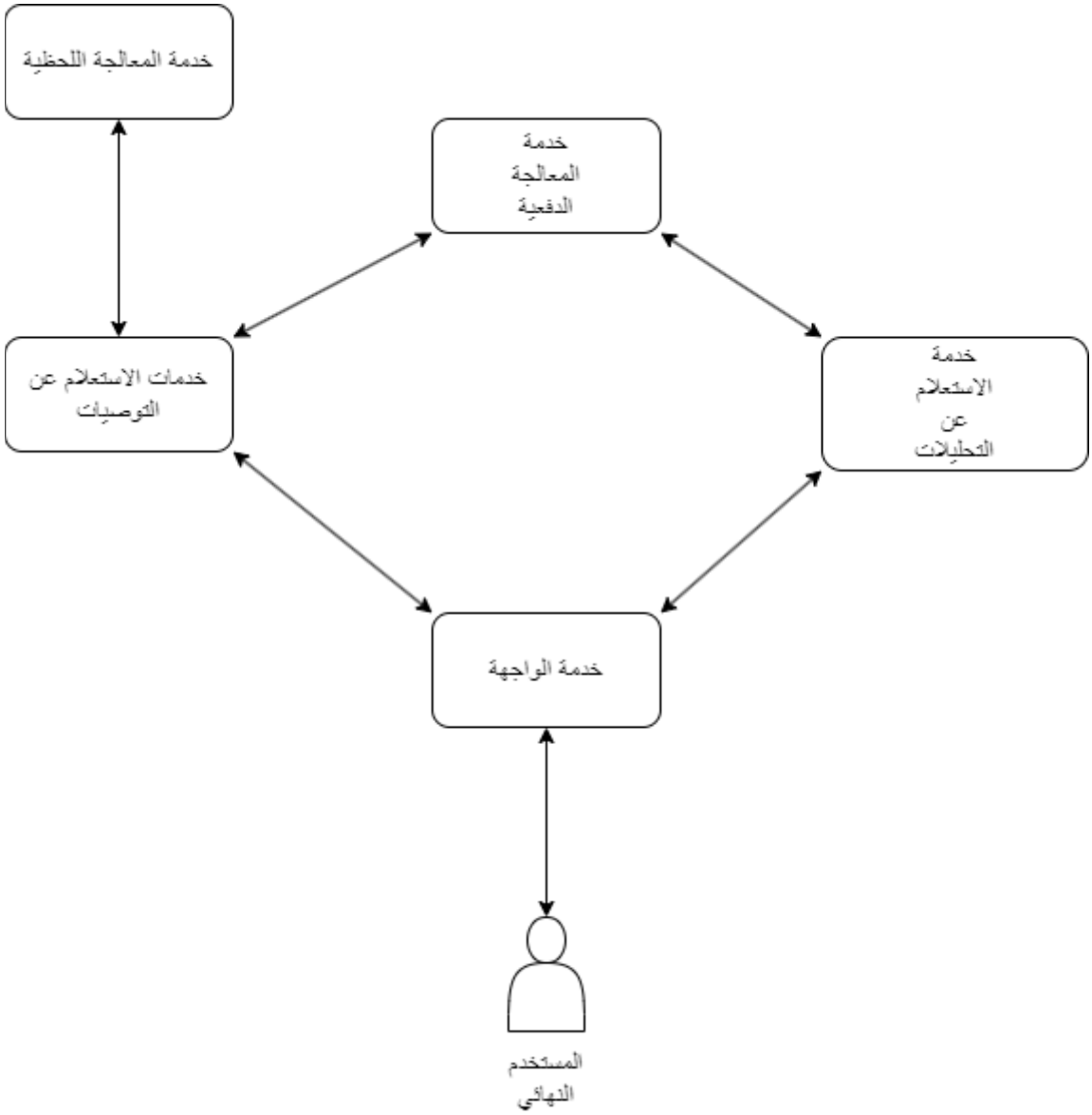
تم استخدام منهجية البنية سداسية الشكل (hexagonal architecture).

6.3- خدمة واجهة المستخدم

تم بناء واجهة المستخدم باستخدام بنية قائمة على المكونات مما يوفر العديد من الفوائد أهمها:

- النسقية (Modularity): تسمح بتقسيم موقع الويب إلى وحدات أصغر (مكونات) مستقلة. يغلف كل مكون وظيفة محددة، مما يجعل إدارته وفهمه أسهل.
- قابلية إعادة الاستخدام: بمجرد إنشاء أحد المكونات، يمكن إعادة استخدامه عبر أجزاء مختلفة من موقع الويب أو حتى في مشاريع مختلفة. يقلل هذا من التكرار ويسرع التطوير ويضمن الاتساق في واجهة المستخدم والسلوك.
- سهولة تصحيح الأخطاء: تعمل المكونات المعزولة على تبسيط عملية تصحيح الأخطاء، حيث يمكن تتبع المشكلات إلى مكونات معينة بدلاً من البحث ضمن قاعدة الرماز الكبيرة (huge codebase).

7.3- مخطط النظام التصميمي



صورة **2** مخطط يوضح كيفية التواصل بين خدمات النظام

الفصل الرابع

تنجيز النظام

يعرض هذا الفصل كيفية تنجيز النظام مع تفصيل كل جزء من أجزائه.

1.4- مقدمة

تم اتباع منهجية تطوير حديثة تضمن جودة الكود، سهولة التعاون، وأتمتة عمليات النشر. تم رفع الرماز المصدري الخاص بكل خدمة باستخدام نظام التحكم بالإصدارات Git إلى منصة GitHub على شكل فروع. هذا الأسلوب يسمح بتتبع التغييرات بدقة ويتيح العمل على ميزات جديدة في فروع معزولة (feature branches) قبل دمجها في الفرع الرئيسي للتطوير (dev).

لتحقيق التكامل والنشر المستمر (CI/CD)، تم استخدام خادم الأتمتة Jenkins. تم إنشاء خط أنابيب مؤتمت (Pipeline) لكل خدمة يعمل بشكل تلقائي عند دمج التغييرات في الفرع الرئيسي (master). يقوم خط الأنابيب بتنفيذ الخطوات التالية:

1. بناء (Build): تحميل الرماز المصدري من GitHub وتشغيل أداة البناء Maven لتجميع الكود وإنتاج ملفات جافا التنفيذية (JAR files).
2. اختبار (Test): تشغيل مجموعة الاختبارات الآلية (Unit & Integration Tests) للتأكد من صحة التغييرات.
3. تعبئة (Package): بناء صور Docker للخدمات التي سيتم نشرها كحاويات (مثل خدمات API).
4. نشر (Deploy): دفع صور Docker إلى سجل الحاويات (Container Registry) ونشر مهام Spark/Flink أو تطبيق تحديثات Kubernetes على بيئة التشغيل.

تضمن هذه المنهجية تسريع وتيرة التطوير وتقليل الأخطاء البشرية والحفاظ على بيئة نشر مستقرة ومتوقعة.

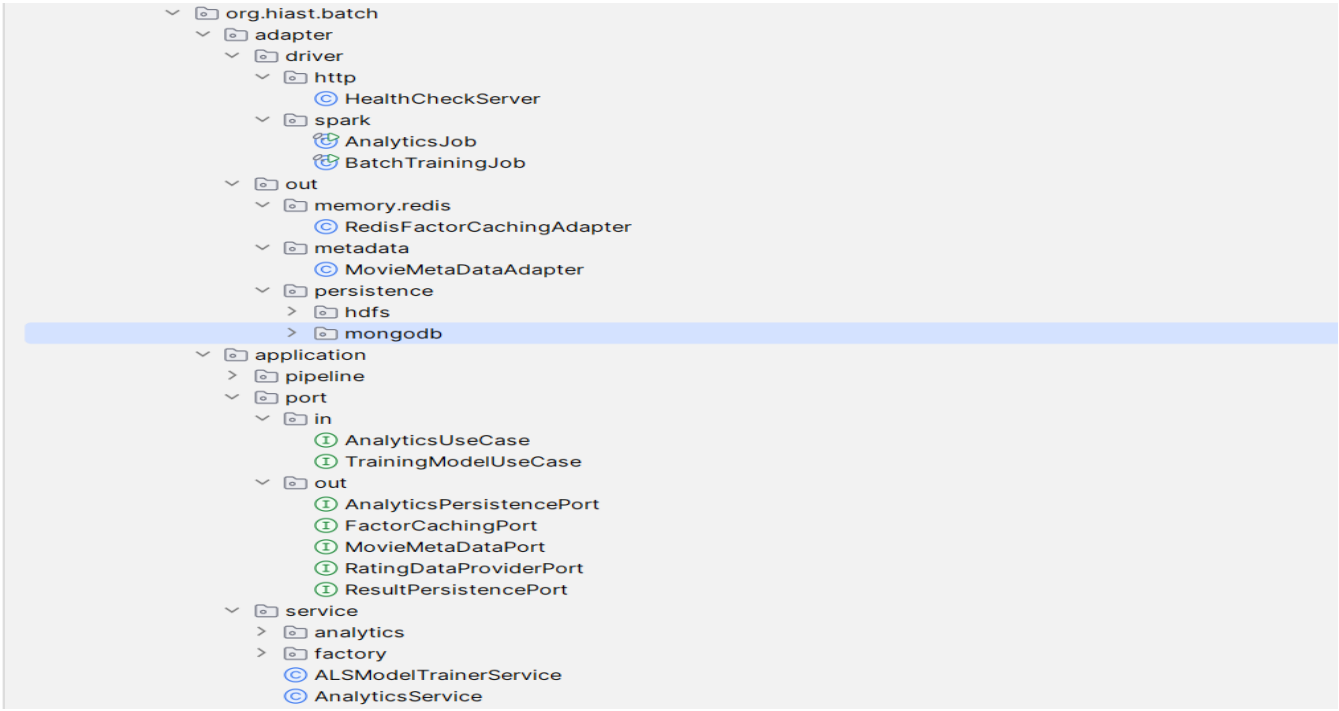
2.4- تنجيز خدمة المعالجة الدفعية

1.2.8- مقدمة

تم بناء هذه الخدمة الموحدة باستخدام لغة Java وإطار عمل Apache Spark. تم اتخاذ قرار دمج مسؤوليتي تدريب النموذج وتحليل البيانات في هذه الخدمة لتحقيق أقصى استفادة من الموارد وإعادة استخدام **كود تحميل ومعالجة البيانات**، حيث تعمل كلتا المهمتين على نفس مجموعة بيانات MovieLens[1] [18].

2.2.8- تفاصيل التنجيز

تم تنجيز الخدمة وفقاً لمبادئ البنية السداسية (Hexagonal Architecture) لعزل منطق العمل عن التفاصيل التقنية.



صورة **3** تنجيز خدمة المعالجة الدفعية

[1] تم استخدام مجموعة بيانات تحوي على 33 مليون تقييم، وهي أكبر مجموعة بيانات لها معنى (اسم الفيلم ونوعه وخصائصه) تم إيجادها، إذ توجد بيانات بحجوم أكبر لكن تتضمن معرف المستخدم ومعرف الفيلم والتقييم فقط، وهذا غير مناسب لعملية التوصية التي نقوم بها، إذ سنقوم بتلك الحالة بإجراء توصية دون معرفة خصائص هذه التوصية وعلى ماذا تعبر، أي لا تخدم مفهوم التنوع في بيئات المعطيات الكبيرة.

نواة المشروع (Core):

- طبقة النطاق (Domain): تحتوي على تعريفات كائنات Java البسيطة (POJOs) التي تمثل المفاهيم الأساسية مثل UserFactor, ItemFactor, هذه الطبقة لا تعتمد على أي مكتبات خارجية سوى النواة المشتركة.

- طبقة التطبيق (Application): تحتوي على منطق العمل. تم تعريف واجهات المنافذ (port) مثل HdfsRatingDataProviderAdapter (لتحميل البيانات)، FactorCachingPort (لحفظ العوامل في Redis)، وAnalyticsPersistencePort (لحفظ التحليلات في MongoDB). وتقوم خدمات مثل ALSModelTrainerService وAnalyticsService بتنفيذ هذا المنطق.

طبقة المحولات (Adapters):

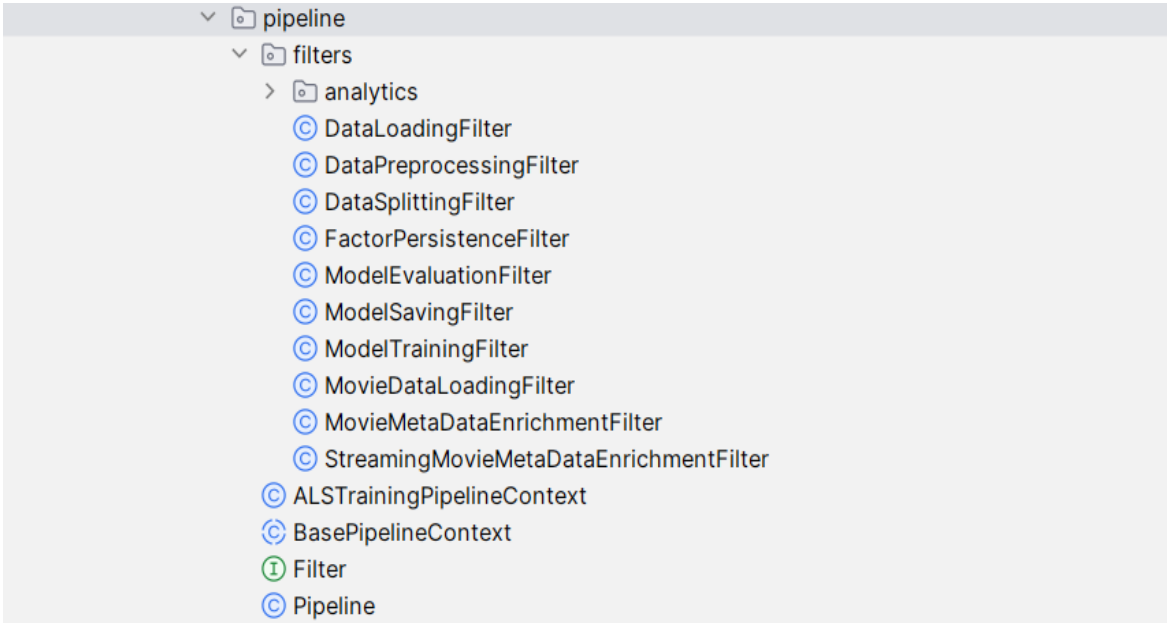
- محول التشغيل (Driver): مثل تعريف BatchProcessingJob التي تحتوي على الدالة main. تعمل كنقطة دخول لمهمة Spark، حيث تقوم بتهيئة SparkSession، وإنشاء تطبيقات المحولات (مثل محول Redis ومحول HDFS)، وتنسيق استدعاء خدمات التطبيق.
- محولات الإخراج (Driven): هي تطبيقات ملموسة للمنافذ (تنجيزات). على سبيل المثال، يقوم RedisFactorCachingAdapter بتنفيذ منطق تحويل متجهات العوامل إلى بايتات ([] byte) وتخزينها في Redis، بينما يقوم MongoAnalyticsPersistenceAdapter باستخدام موصل Spark-MongoDB لكتابة DataFrames التي تحتوي على نتائج التحليل إلى MongoDB.

3.2.4- مسار تدريب النموذج

عند تشغيل مهمة التدريب، تقوم الخدمة بالخطوات التالية:

1. تحميل بيانات التقييمات والأفلام منHDFS
2. تقسيم البيانات إلى مجموعة تدريب واختبار.
3. تدريب نموذج ALS باستخدام المعاملات الفائقة المحددة في ملفات الإعدادات.
4. تقييم النموذج باستخدام مقياسRMSE
5. استخراج عوامل المستخدمين والأفلام وتخزينها فيRedis

6. حفظ نسخة من التوصيات الدفعية لكل مستخدم فيMongoDB



صورة **4** مسارات المعالجة الدفعية

4.2.4- مسار تحليل البيانات

عند تشغيل مهمة التحليل، تقوم الخدمة بالخطوات التالية:

1. تحميل بيانات التقييمات والأفلام والوسوم من HDFS.
2. تنفيذ مجموعة من الاستعلامات التحليلية باستخدام Spark SQL (مثل حساب متوسط التقييمات لكل نوع، إيجاد الأفلام الأكثر تقييماً، إلخ).
3. تجميع النتائج في DataFrames.
4. كتابة كل DataFrame يمثل نتيجة تحليل إلى مستند (document) منفصل في MongoDB.

صورة **5** مسار تحليل البيانات

(تم تجميع فلاتر تحليل البيانات بناءً على القدرة الحسابية اللازمة لإنجاز كل تحليل لأنه في البداية تم مواجهة عدة اختناقات في الأداء خلال هذا المسار فتم الفصل بناءً على ذلك لمعرفة نقطة الاختناق آنذاك).

3.4- تنجيز خدمة المعالجة اللحظية

1.3.4- مقدمة

تم بناء هذه الخدمة باستخدام لغة Java وإطار عمل Apache Flink. تم اختيار Flink لقدرته على معالجة الأحداث بزمن استجابة منخفض جداً ودعمه المتقدم لإدارة الحالة، وهو أمر ضروري لتوفير توصيات لحظية ذات جودة عالية.

2.3.4- تفاصيل التنجيز

تم اتباع البنية السداسية أيضاً في هذه الخدمة.

نواة المشروع (Core): تحتوي على منطق العمل الأساسي، بما في ذلك خدمة RealtimeRecommendationService التي تنسق عملية التوصية اللحظية، والمنافذ التي تعزلها عن التفاصيل الخارجية مثل VectorSearchPort.

طبقة المحولات (Adapters):

محول التشغيل (Driver): مثل RealTimeRecommendationsJob التي تعرف خط أنابيب Flink. تعمل ك "محول إدخال" من خلال استهلاك الأحداث من Apache Kafka.

محولات الإخراج (Driven): هي تطبيقات للمنافذ، مثل VectorSearchAdapter يقوم ببناء وتنفيذ استعلامات البحث عن المتجهات (KNN) باستخدام FT. SEARCH.

3.3.4- آلية المعالجة اللحظية

1. تستهلك مهمة Flink حدث تفاعل مستخدم من موضوع Kafka.
2. لكل حدث، يتم استدعاء خدمة RealtimeRecommendationService.
3. تقوم الخدمة أولاً بجلب متجه عوامل الفيلم الذي تم تقييمه من Redis.

4. ثم تستخدم متجه الفيلم كـ "متجه استعلام" لتنفيذ بحث متجهي في Redis (RediSearch) عن أقرب متجهات عوامل الأفلام.
5. تستقبل قائمة بالأفلام المرشحة المخصصة للمستخدم.
6. تقوم بنشر قائمة التوصيات النهائية إلى موضوع Kafka آخر.

لتحقيق أقصى أداء، يتم استخدام إطار عمل التسلسل Apache Fury لتحويل كائنات Java التي يتم تمريرها بين مهام Flink أو إرسالها عبر Kafka إلى تمثيل ثنائي مضغوط، مما يقلل من حمل الشبكة ويزيد من سرعة المعالجة.

4.4- تنجيز خدمات الاستعلام

1.4.4- مقدمة

لتوفير التوصيات ونتائج التحليل للتطبيقات الأخرى، تم تصميم خدمتين API مستقلتين باستخدام Spring Boot وJava.

2.4.4- خدمة الاستعلام عن التوصيات

هذه الخدمة تقدم واجهة REST API لاسترجاع التوصيات. تحتوي على نقاط نهاية (Endpoints) مثل /api/recommendations/{userId}. عند استدعائها، تقوم بالاستعلام من MongoDB لجلب التوصيات الدفعية المعدة مسبقاً، أو يمكن تطويرها لتستمع لموضوع Kafka الخاص بالتوصيات اللحظية وتوفرها للمستخدمين.

3.4.4- خدمة الاستعلام عن التحليلات

تقدم هذه الخدمة واجهة REST API لاسترجاع نتائج التحليل. تحتوي على نقاط نهاية. عند استدعائها، تقوم بالاستعلام من MongoDB لجلب البيانات المجمعة من المجموعة المناسبة، وتنسيقها بصيغة JSON، وإعادتها لواجهة المستخدم.

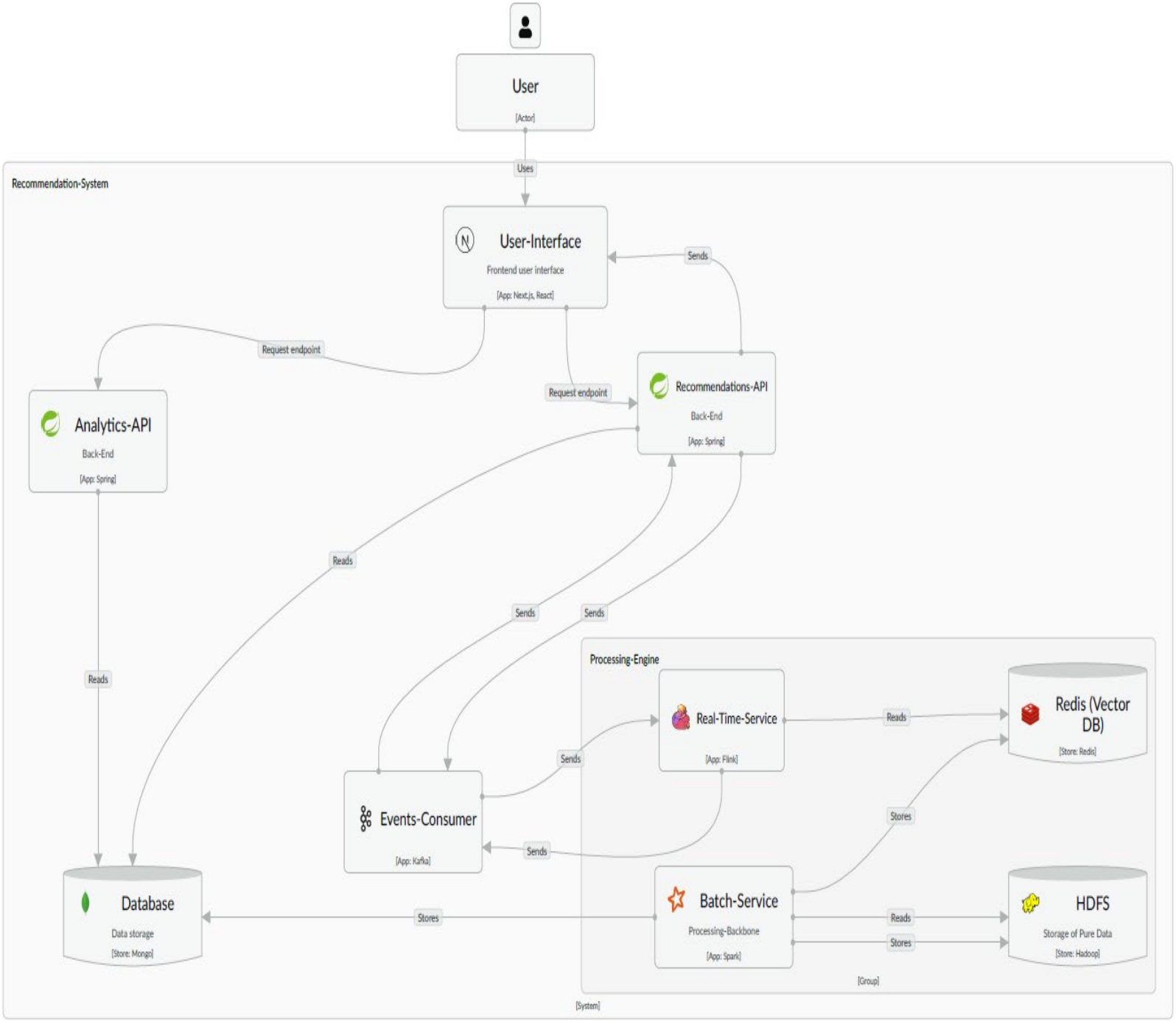
5.4- تنجيز واجهة المستخدم (Dashboard)

تم بناء واجهة المستخدم كـ "لوحة تحكم تحليلية" باستخدام إطار العمل Next.js ومكتبة React. تم اتخاذ هذا القرار للاستفادة من قدرة Next.js على بناء واجهات تفاعلية سريعة وحديثة.

آلية العمل: تقوم لوحة التحكم باستدعاء خدمة الاستعلام عن التحليلات لجلب بيانات التحليل، وتقدم واجهة تجريبية بسيطة لاختبار خدمة الاستعلام عن التوصيات وخدمة المعالجة اللحظية.

العرض: يتم عرض تحليلات البيانات على شكل رسوم بيانية تفاعلية (مثل المخططات الشريطية والدائرية) لتسهيل فهم الرؤى المستخلصة على محللي البيانات.

7.4- مخطط النظام التنجيزي



صورة **6** كيفية التواصل بين الخدمات ضمن النظام مع الأدوات المستخدمة.

8.4- نشر النظام

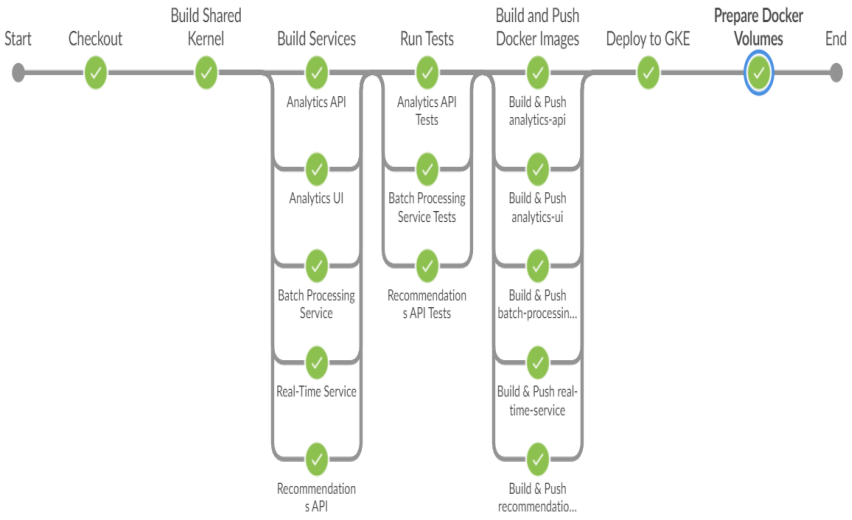
تم تجهيز البنية التحتية لنشر الخدمات عن طريق استخدام عنقود Kubernetes على منصة GKE، حيث يتم نشر الخدمات على هذا العنقود باستخدام آلة افتراضية منفصلة مثبت عليها Jenkins

من أجل أتمتة عملية نشر الخدمات، تم إنشاء خط أنابيب مؤتمت (Pipeline) باستخدام الأداة Jenkins، يعمل بشكل تلقائي عند حدوث تعديل ضمن الفرع master الخاص بمستودع الخدمة، إذ يتم تحميل الرماز المصدري الخاص بكل خدمة وبناؤه ضمن صورة Docker ثم رفع الصورة إلى Docker hub بعد ذلك يتم تحميل هذه الصور من العنقود وانشاء الحاويات الموافقة لكل خدمة.

من خلال اعتماد هذه المنهجية في نشر التطبيق تم تحقيق مجموعة من الميزات أهمها:

- الحفاظ على الرماز البرمجي من الضياع وإمكانية مشاركته ضمن فريق العمل بسهولة، حيث تم تحقيق ذلك من خلال استخدام الأداة Git والموقع GitHub.
- عدم الحاجة إلى إعادة بناء الخدمات بشكل يدوي عند تعديلها ضمن المستودعات، إذ يعمل Jenkins على أتمتة هذه العملية، مما يسرع عملية التطوير والنشر ويحقق هذا مبدأ (Continues Delivery).

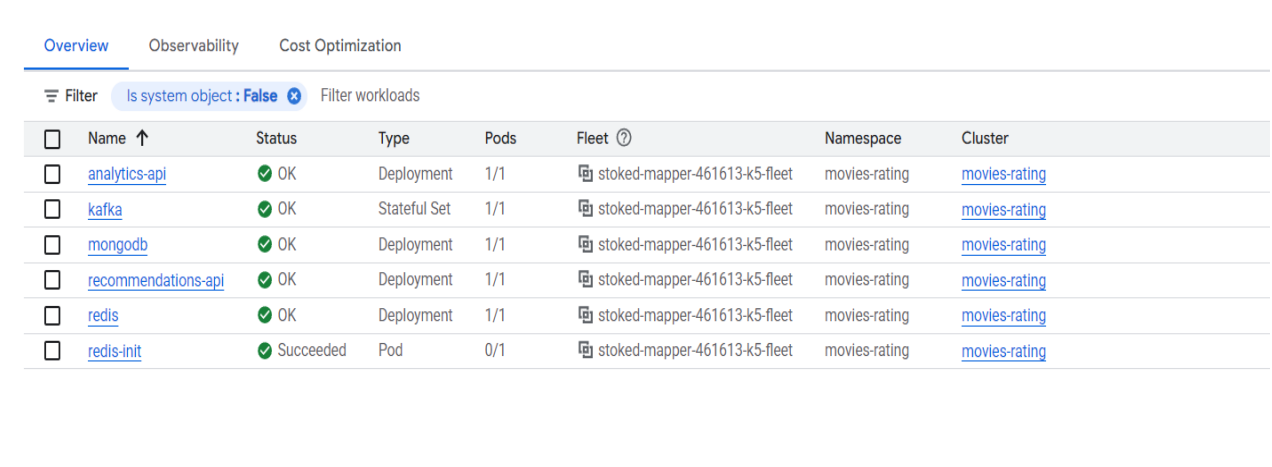
عدم الحاجة إلى إدارة الاعتماديات الخاصة بكل خدمة ودراسة تأثيرها على الخدمات الأخرى، حيث إن استخدام Docker يساعد على إنشاء حاوية معزولة وخاصة لكل خدمة تحتوي على الخدمة مع جميع تبعياتها.



صورة 7 خط أنبوب **Jenkins**

تظهر الصورة خط الأنابيب (Pipelines) المنجز باستخدام Jenkins التي تفعل مباشرة عند تعديل الفرع main ضمن المستودع الخاص بالخدمة، كما يمكن تشغيلها يدوياً من الواجهة.

بعد انتهاء هذا الخط من التنفيذ تكون الخدمات قد نشرت على العنقود كما في الصورة:



صورة 8 عنقود **GKE**

نلاحظ من الصورة انتهاء نشر الخدمات إذ أصبحت جاهزة للعمل (إذ تم نشر خدمة الواجهة محلياً بسبب نقص الموارد التي يمكن اسنادها لعنقود GKE الخاص بنا).

إضافة للعنقود على منصة GKE تم انشاء عنقود آخر لعمليات المعالجة على منصة Google DataProc المختصة في نمط معالجات أو أحمال البيانات الضخمة لعزل خدمات الحوسبة عن خدمات التخزين والاستعلام لأننا بحاجة لمنصة عالية الفعالية لمعالجة هذه الأحمال بشكل أفضل من نشر تلك الخدمات على عنقود GKE

تظهر الصورة التالية عناقيد DataProc المستخدمة (تم انشاء أكثر من عنقود لأغراض الاختبار).

Name	Status	Region	Zone	Total worker nodes	Flexible VMs?	Scheduled deletion
movies-rating	Stopped	us-central1	us-central1-a	0	No	Off
movies-rating-three-w-nodes	Stopped	us-east1	us-east1-b	3	No	Off
movies-rating-two-w-nodes	Stopped	us-central1	us-central1-b	2	No	Off

صورة 9 عناقيد **DataProc**

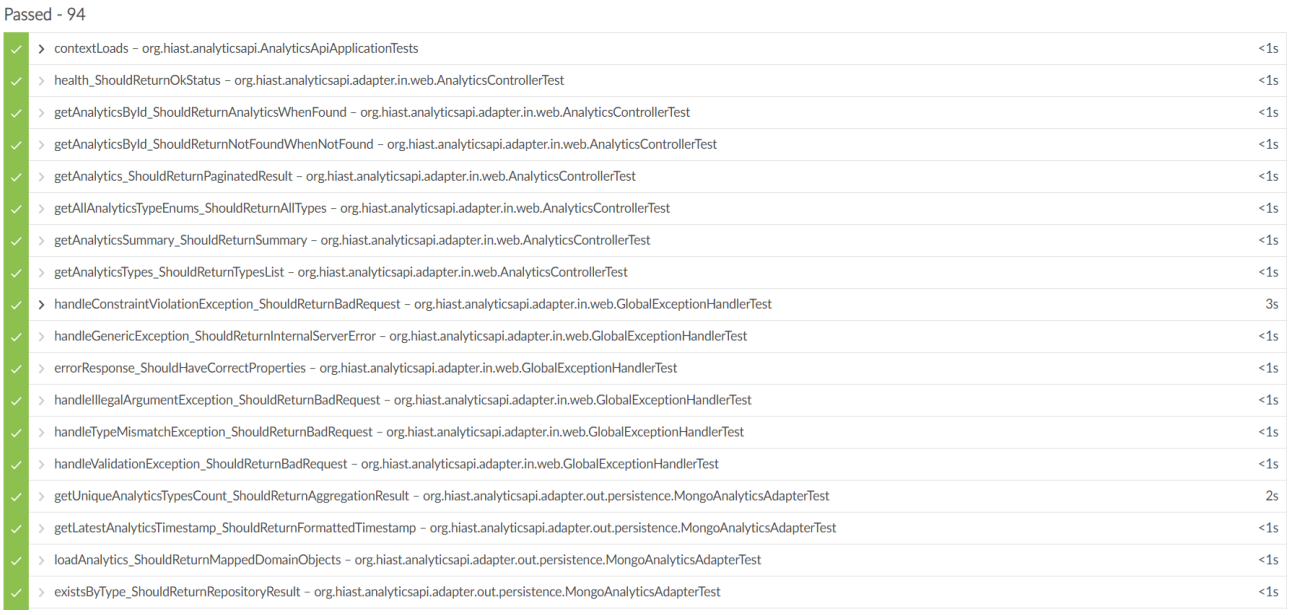
الفصل الخامس

اختبارات النظام ومناقشة النتائج

يوضح هذا الفصل الاختبارات التي تم تطبيقها للتأكد من تلبية النظام للمتطلبات.

1.5- اختبار الوحدات

إن أهمية مثل هذه الاختبارات تكمن في كونها تركز على المستوى الأدنى من التطبيق، حيث نستطيع التأكد من أن الخدمات، الكيانات، التجميعات وأغراض القيمة ضمن طبقة المجال تحقّق المطلوب قبل الغوص في مشاكل الربط مع قواعد المعطيات والمكاتب الخارجيّة.



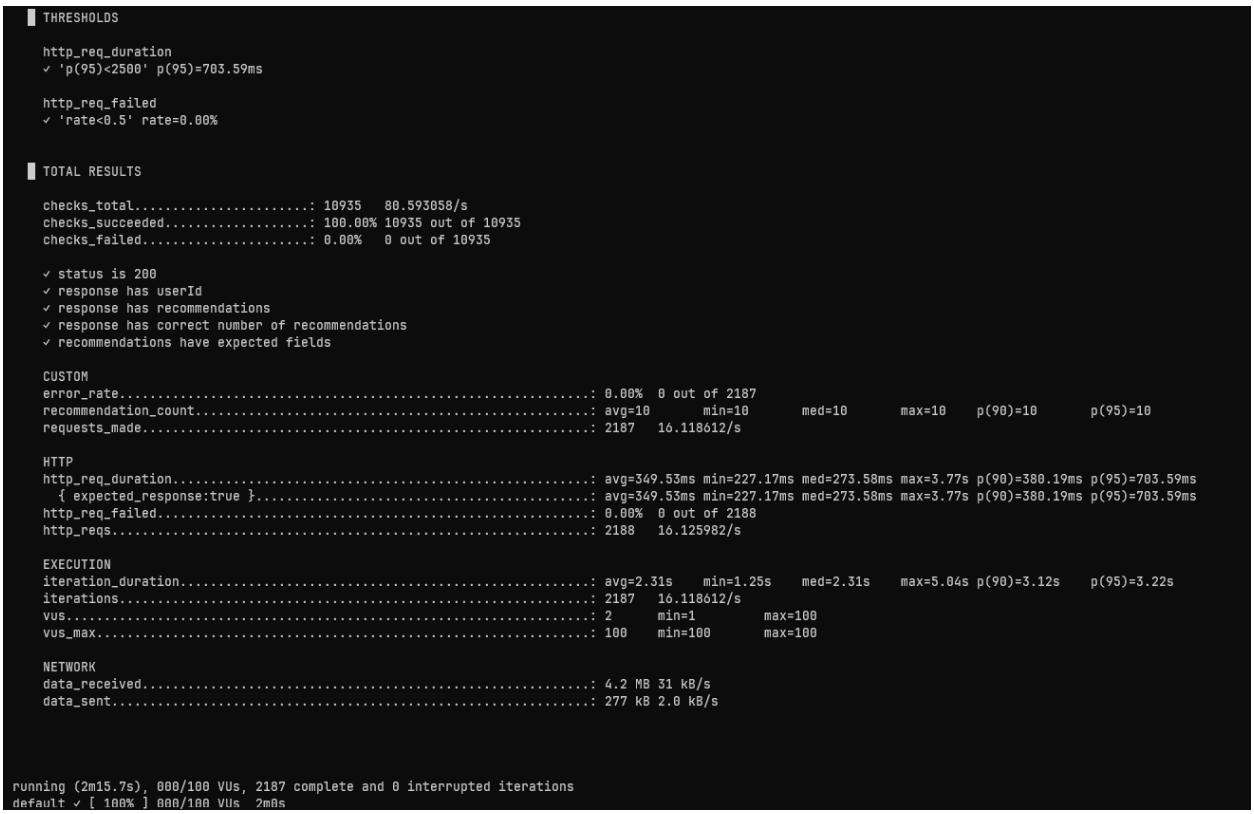
صورة 10 اختبارات الوحدة

تبين الصورة اختبارات الوحدة التي كتبت للخدمات، والتي يشغلها خط أنابيب Jenkins بشكل مؤتمت.

2.5- اختبار خدمة الاستعلام عن التوصيات

من أجل التأكد من قدرة خدمة التخزين على الاستجابة لعدد كبير من الطلبات، تم إنشاء اختبارات لأداء هذه الخدمة باستخدام الأداة k6، سيتم استعراض كيفية التجهيز للاختبارات مع نتائجها في الفقرات التالية.

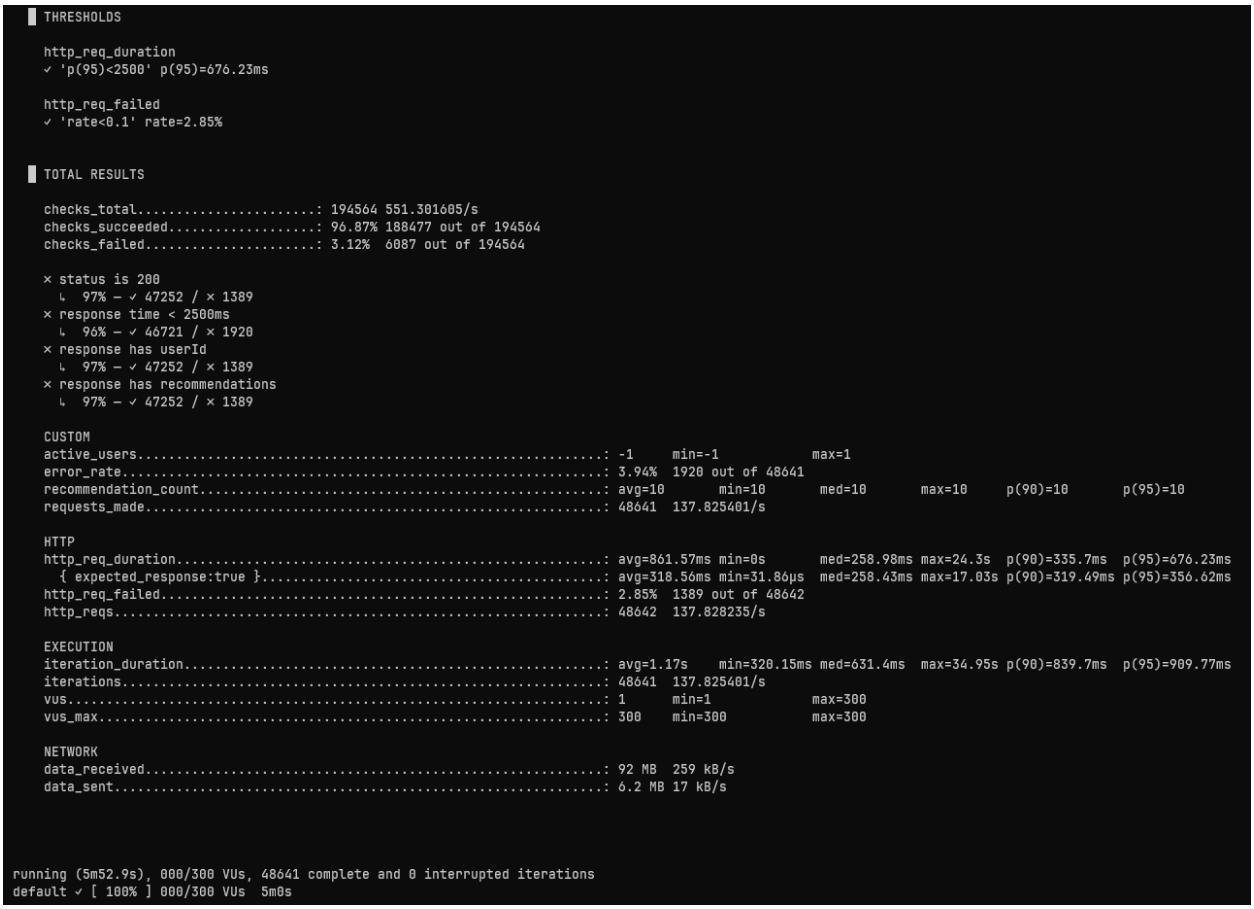
1.2.5- اختبار الحمل



صورة 11 اختبار الحمل لخدمة الاستعلام عن التوصيات

نلاحظ خلال الاختبار عدم فشل أي طلب من الطلبات الناتجة عن 100 مستخدم يستعلم عن التوصيات بزمن تأخر للطلبات ((P(95) 703 ميلي ثانية وهذه نتيجة ممتازة مقارنة بقوة الموارد التي تستخدمها الخدمة على GKE والتي تعمل على 1vCPU بالحد الأدنى و 2vCPU بالحد الأعلى.

2.2.5- اختبار الضغط



صورة 12 اختبار الضغط لخدمة الاستعلام عن التوصيات

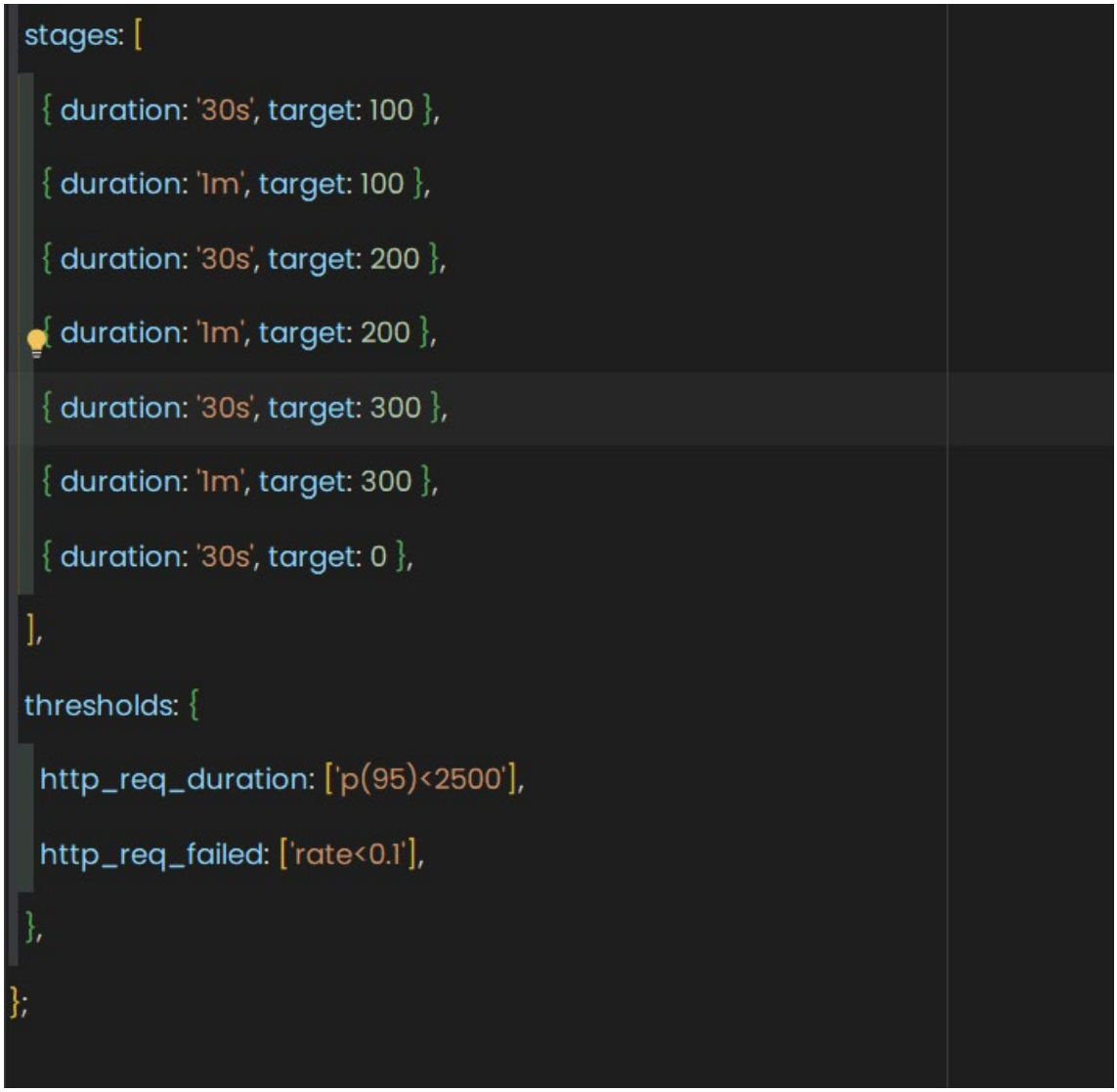
نلاحظ من الاختبار أن هناك عددا من الطلبات قد فشل ونسبتهم حوالي 3% وذلك لأننا وضعنا شرط مسبق على الطلبات وهو في حال تجاوز تأخر الطلب P(95) لأكثر من 2500 ميلي ثانية فإنه يعتبر طلب فاشل، ولكن بالعموم نتيجة هذا الاختبار ممتازة إذ يخدم 300 مستخدم على التوازي لمدة دقيقتان على موارد حوسبة متواضعة كما ذكرنا في الاختبار السابق. إذ يعزى هذا الأداء إلى كفاءة الاستعلام من قاعدة بيانات MongoDB وتكاملها مع Spring Boot المناسبة لعمليات القراءة السريعة. كما يوضح أن الموارد المخصصة على GKE (1-2 vCPU) كانت كافية لهذا الحمل.

3.2.5- ملاحظات حول طبيعة الاختبار

طبيعة اختبار الحمل:

صورة 13 معاملات اختبار الحمل

طبيعة اختبار الضغط:



صورة 14 معاملات اختبار الضغط

3.5- اختبار خدمة المعالجة الدفعية

لتقييم أداء وكفاءة هذه الخدمة، تم إجراء اختبار معياري يهدف إلى مراقبة استهلاك الموارد خلال تنفيذ مهمة وتحليل البيانات على مجموعة بيانات MovieLens الكاملة.

1.3.5- بيئة وهدف الاختبار

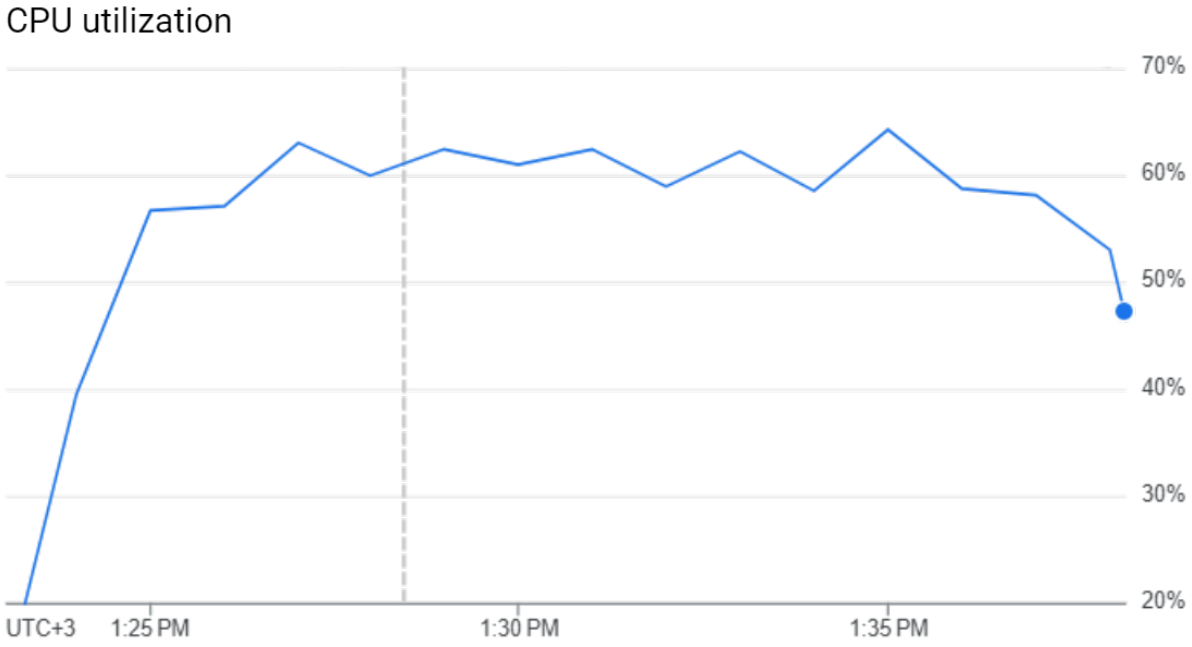
- **البيئة :**تم تنفيذ الاختبار على عنقود **Google Cloud Dataproc** تم تكوينه للعمل كعقدة وحيدة-Single Node Cluster، مما يعني أن عقدة السيد (Master Node) تقوم بجميع مهام المعالجة.

- **مواصفات العقدة :**تم استخدام آلة افتراضية من نوع n4-standard-8 ، والتي توفر 8 وحدات معالجة مركزية افتراضية (vCPUs) و32 جيجابايت من ذاكرة الوصول العشوائي.

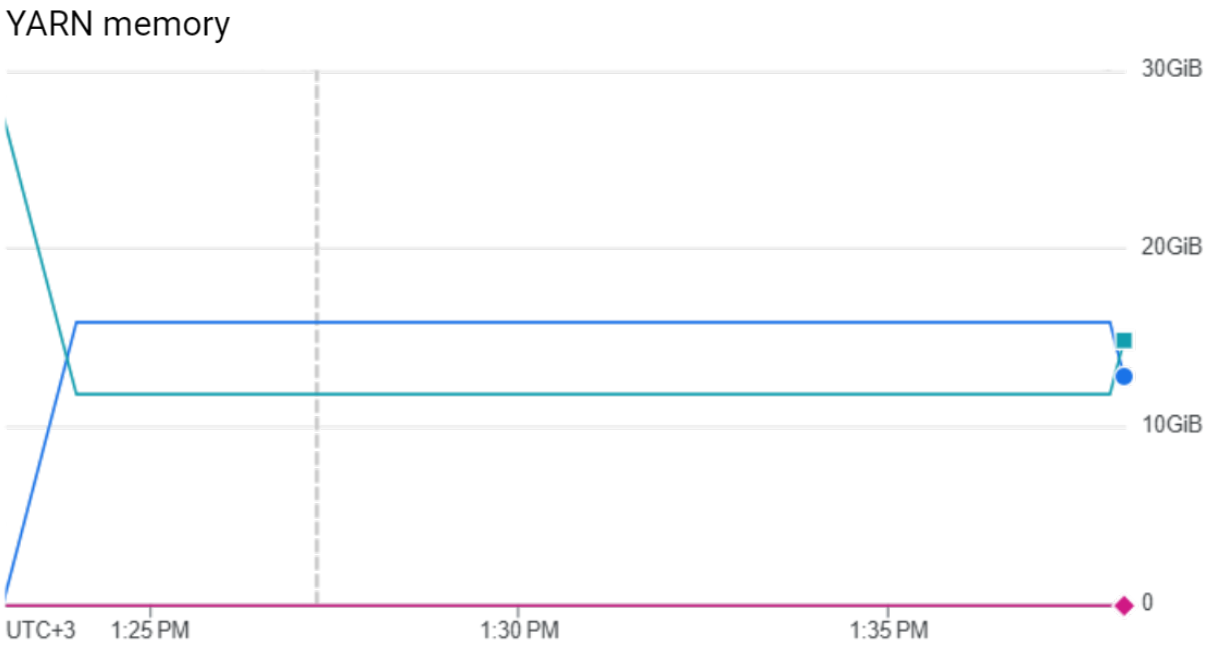
الهدف :الهدف من هذا الاختبار هو قياس الأداء الأساسي (Baseline Performance) للخدمة ومراقبة كيفية استفادتها من الموارد المتاحة في بيئة محكومة، وذلك لتوفير أساس للمقارنة عند التوسع إلى عناقيد متعددة العقد.

2.3.5- تحليل النتائج ومراقبة الموارد

استغرقت مهمة Spark ما يقارب **15 دقيقة** لإكمال التنفيذ على العنقود أحادي العقدة. خلال هذه الفترة، تم تسجيل مقاييس استهلاك الموارد من واجهة مراقبة YARN



صورة 15 استهلاك المعالج للعنقود ذي العقدة الواحدة



صورة 16 استهلاك الذاكرة للعنقود ذي العقدة الواحدة

3.3.5- الاستنتاج

تُظهر نتائج الاختبار أن هذه الخدمة تعمل بكفاءة عالية على البنية التحتية المحددة. إن الاستخدام المستقر والمرتفع لكل من الذاكرة والمعالج يشير إلى أن تطبيق Spark مُحسَّن بشكل جيد ولا يعاني من اختناقات في الموارد على هذا النطاق. توفر هذه النتائج خط أساس قوياً يؤكد على صحة التنجيز، وتفتح المجال لإجراء اختبارات توسع مستقبلية.

4.5- اختبار قابلية التوسع لخدمة المعالجة الدفعية

لتقييم قدرة النظام على التوسع الأفقي (Horizontal Scaling)، تم تنفيذ نفس مهمة المعالجة الدفعية على عنقودين إضافيين بتكوينات مختلفة، مع زيادة عدد العقد العاملة (Worker Nodes).

1.4.5- بيئات الاختبار الإضافية

تم استخدام عنقودين إضافيين على **Google Cloud Dataproc** بالإضافة إلى العنقود الأساسي:

- **العنقود الثاني** :عقدة سيد واحدة + **عقدتين عاملتين (2 Workers)** ، كل عقدة من نوع n4-standard-2 2vCPU, 8 GB RAM
- **العنقود الثالث** :عقدة سيد واحدة + **ثلاث عقد عاملة (3 Workers)** ، كل عقدة من نوع n4-standard-2 2 vCPU, 8 GB RAM

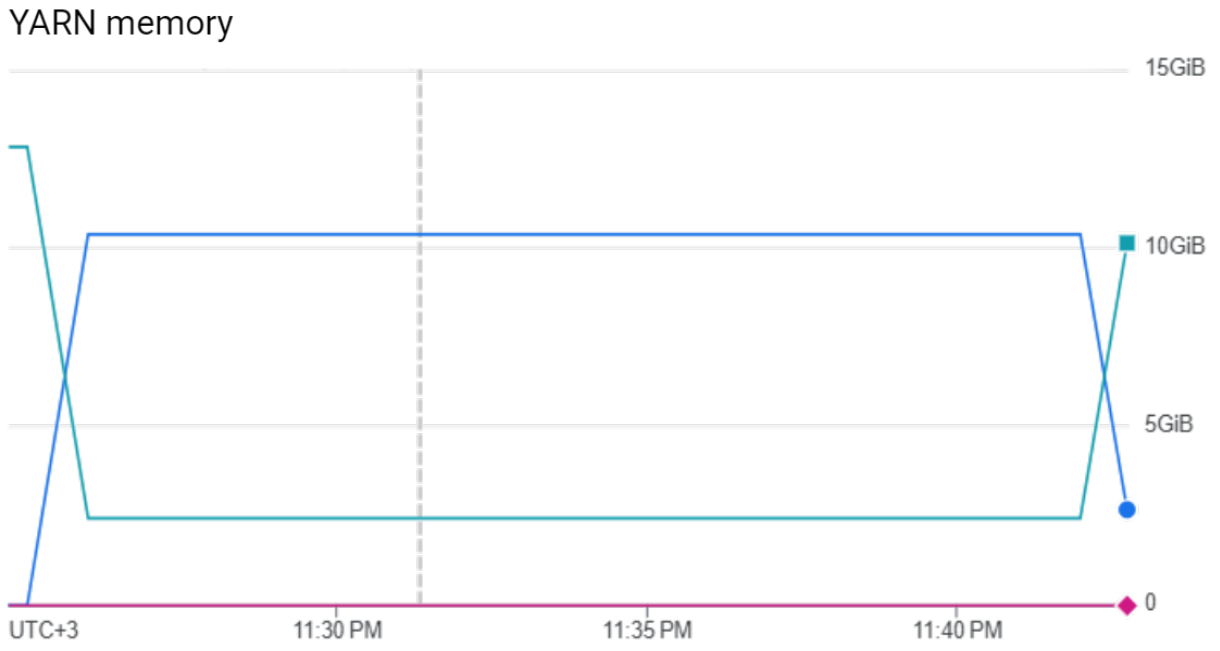
الهدف :قياس تأثير توزيع حمل العمل على عدة عقد عاملة على زمن التنفيذ الكلي واستهلاك الموارد، والتحقق من قدرة النظام على الاستفادة من الموارد الإضافية.

2.4.5- تحليل نتائج العنقود ذي العقدتين العاملتين (2 Workers)

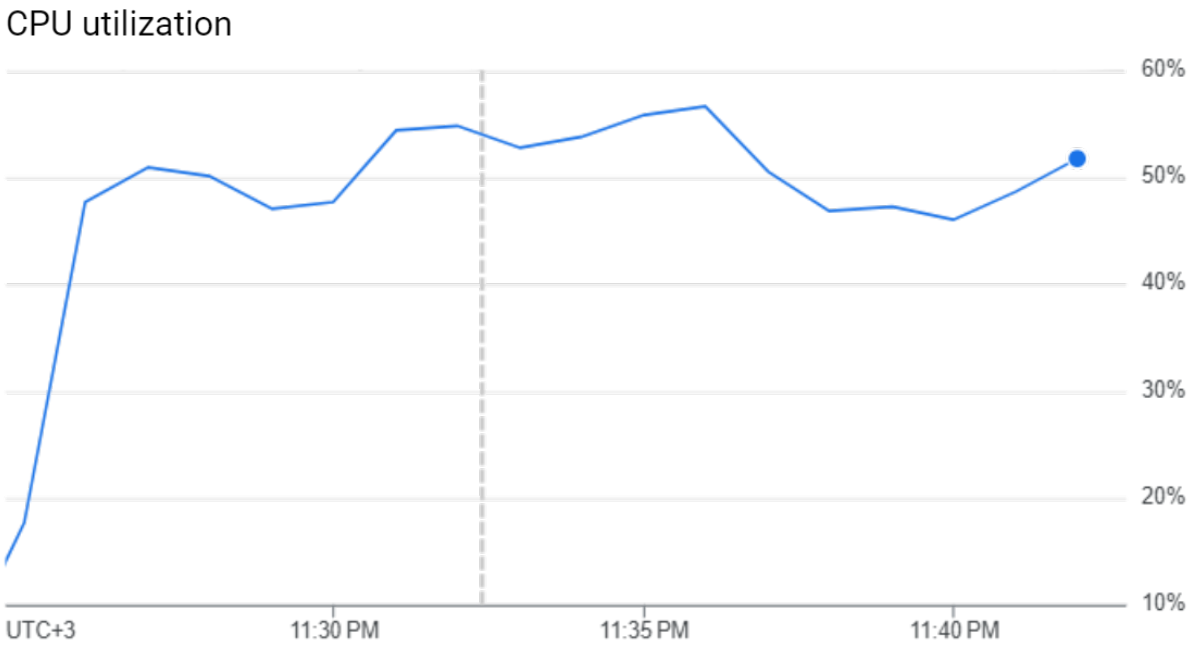
استغرقت المهمة على هذا العنقود ما يقارب **18 دقيقة** لإكمال التنفيذ.

- **استخدام الذاكرة** :يوضح الشكل أدناه أن الذاكرة المخصصة في YARN استقرت عند حوالي **10 جيجابايت**، وهو ما يتناسب مع الموارد المتاحة في العقد العاملة.

استخدام المعالج :يوضح الشكل أدناه أن متوسط استخدام المعالج كان مرتفعاً ومستقراً، ويتراوح بين **50%و55%**، مما يدل على توزيع جيد للمهام على أنوية المعالجات المتاحة.



صورة 17 استهلاك الذاكرة للعنقود ذي العقدتين



صورة 18 استهلاك المعالج للعنقود ذي العقدتين

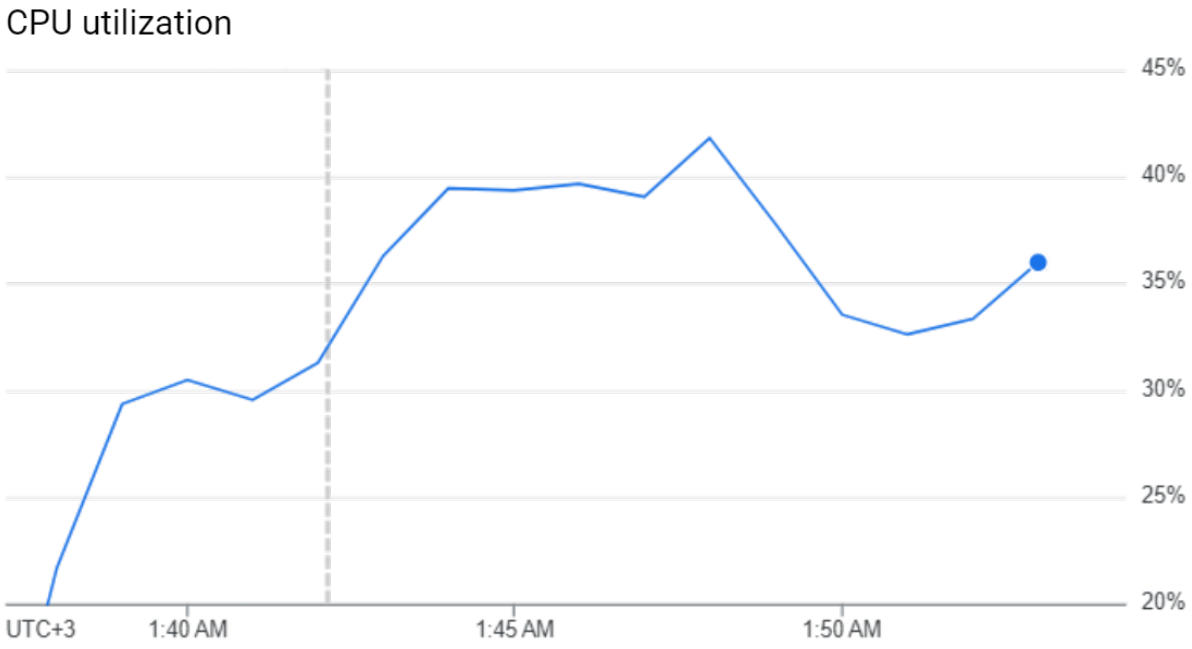
3.4.5- تحليل نتائج العنقود ذي الثلاث عقد عاملة (3 Workers)

استغرقت المهمة على هذا العنقود ما يقارب **17 دقيقة** لإكمال التنفيذ.

- **استخدام الذاكرة :**يوضح الشكل أدناه أن ذاكرة YARN استقرت عند حوالي **11 جيجابايت**، مع زيادة طفيفة لاستيعاب الموارد الإضافية من العقدة الثالثة.

استخدام المعالج :يوضح الشكل أدناه أن استخدام المعالج كان أقل استقراراً مقارنة بالسيناريوهات الأخرى، حيث تراوح بين **30% و 42%** إذ نلاحظ أن استخدام المعالج كان أقل استقراراً وتناغماً مقارنة بالسيناريوهات الأخرى... يشير هذا إلى انخفاض **كفاءة التوازي (Parallelism Efficiency)** مع زيادة عدد العقد العاملة لنفس حجم العمل، تزداد **التكلفة العامة للتنسيق وخلط البيانات عبر الشبكة** .ونتيجة لذلك، تقضي أنوية المعالج فترات أطول في انتظار وصول البيانات أو انتهاء المهام المتأخرة (Stragglers)، مما يؤدي إلى انخفاض متوسط استخدام المعالج الإجمالي على الرغم من أن زمن التنفيذ الكلي يظل متقارباً.

صورة 19 استهلاك الذاكرة للعنقود ذي الثلاث عقد



صورة 20 استهلاك المعالج للعنقود ذي الثلاث عقد

4.4.5- مقارنة النتائج والاستنتاج

لتلخيص نتائج اختبارات التوسع، يعرض الجدول التالي مقارنة بين أداء المهمة على التكوينات الثلاثة:

التكوين	الموارد الحسابية التقريبية (Workers)	زمن التنفيذ	متوسط استخدام المعالج
عقدة واحدة (Master Only)	8 vCPU, 32 GB RAM	~15دقيقة	~62%
عقدتان عاملتان (2 Workers)	4 vCPU, 16 GB RAM	~18دقيقة	~53%
ثلاث عقد عاملة (3 Workers)	6 vCPU, 24 GB RAM	~17دقيقة	~38%

جدول 7 نتائج اختبارات خدمة المعالجة الدفعية على ثلاث عناقيد

الاستنتاج :تُظهر النتائج ملاحظات هامة في أنظمة الحوسبة الموزعة، وتؤكد أن العلاقة بين الموارد والأداء ليست دائماً خطية.

1. **التكلفة العامة للتوزيع (Overhead) :** أظهر سيناريو العقدتين العاملتين زمن تنفيذ أطول من سيناريو العقدة الواحدة. يعود هذا إلى التكلفة العامة التي يضيفها Spark لإدارة وتنسيق المهام وخلط البيانات Data Shuffling عبر الشبكة. بالنسبة لحجم بيانات MovieLens، كانت هذه التكلفة العامة أكبر من الفائدة المكتسبة من توزيع العمل على عقدتين.
2. **كفاءة التوسع العمودي مقابل الأفقي :**حقق سيناريو العقدة الواحدة القوية (n4-standard-8) أفضل أداء. هذا يوضح أن التوسع العمودي (Vertical Scaling) يمكن أن يكون أكثر كفاءة من التوسع الأفقي للأحمال التي يمكن أن تتسع في ذاكرة آلة واحدة، لأنه يتجنب تكاليف الشبكة والتنسيق.

3. **انخفاض كفاءة التوازي (Parallelism Efficiency) :** يتقارب زمن التنفيذ في سيناريو الثلاث عقد عاملة ليتساوى مع زمن العقدة الواحدة، ولكن مع متوسط استخدام للمعالج أقل بكثير (~38%). يشير هذا إلى انخفاض **كفاءة التوازي** .مع زيادة عدد العقد لنفس حجم العمل، تزداد التكلفة العامة للتنسيق، وتقضي أنوية المعالج فترات أطول في حالة انتظار لوصول البيانات أو انتهاء المهام المتأخرة (Stragglers) ، مما يؤدي إلى انخفاض متوسط استخدام الموارد الإجمالي.

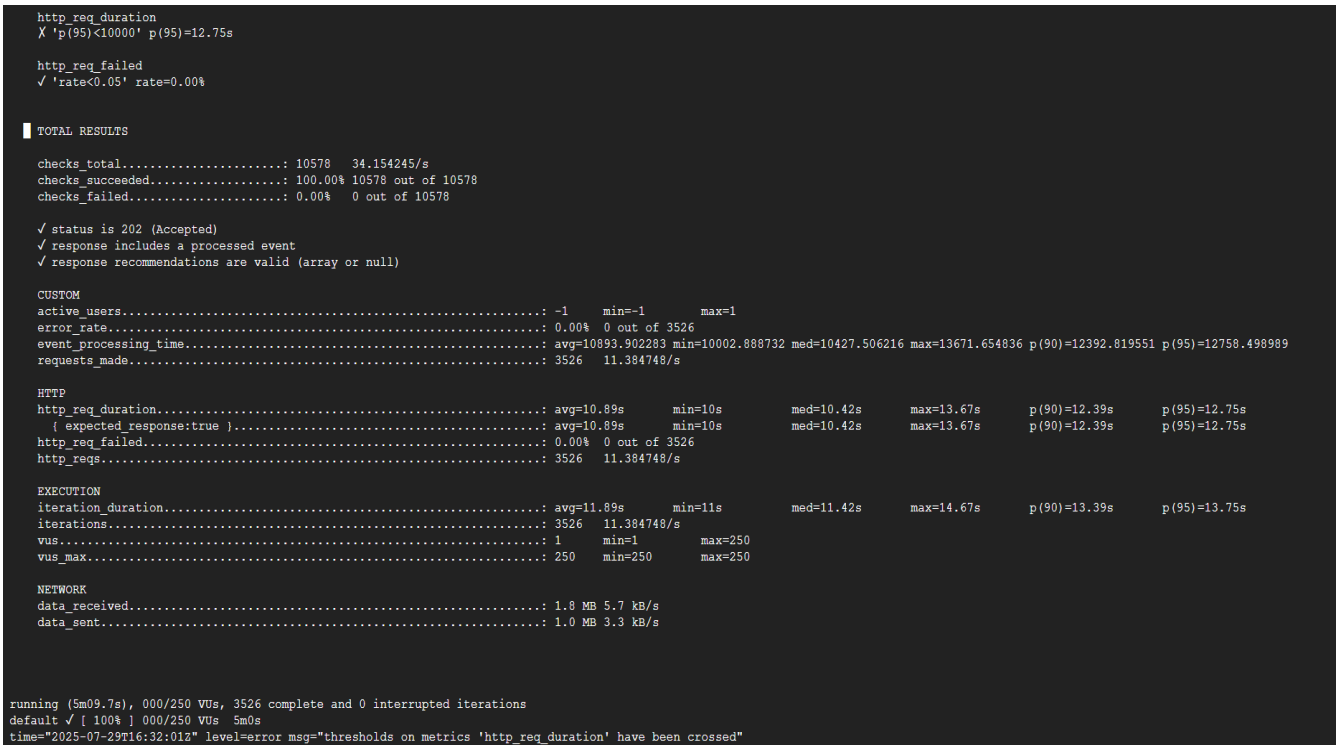
الخلاصة النهائية للاختبار :أثبتت هذه الاختبارات أن هذه الخدمة **قادرة على العمل والتوسع أفقياً** على عناقيد متعددة العقد. ومع ذلك، بالنسبة لحجم البيانات الحالي، فإن التوسع العمودي هو الخيار الأكثر كفاءة. ستظهر الفائدة الحقيقية للتوسع الأفقي بشكل جلي عند معالجة مجموعات بيانات أكبر بكثير (بمقياس التيرابايت)، حيث يصبح من المستحيل احتواء البيانات أو معالجتها على عقدة واحدة، وتصبح التكلفة العامة للتوزيع ضئيلة مقارنة بالمكاسب الهائلة من المعالجة المتوازية.

ملاحظة: تم اختيار مهمة تحليل البيانات بشكل أساسي كآلية تقييم للأداء كونها تتم بوقت قليل نسبياً مقارنة بتدريب النموذج، إذ أن تدريب النموذج مكلف ويحتاج وقت أطول مع نفس طبيعة النتائج.

5.5- اختبار خدمة المعالجة اللحظية

سنكتفي باختبار ضغط هذه الخدمة كون القيود أو الشروط التي يجب أن تحققها هذه الخدمة على أداء عالي. وسيتم الاختبار على عنقود DataProc ذو العقدة الواحدة والتي رأينا في اختبارات المعالجة الدفعية بأن هذا العنقود ذو أعلى استهلاك للموارد الحسابية إضافة لتقليل الحمل الشبكي على الخدمة.

اختبار الضغط:



صورة 21 اختبار الضغط على خدمة المعالجة اللحظية

نلاحظ من نتائج اختبار الضغط أن زمن التأخر P(95) وصل إلى 13 ثانية، وهو زمن مرتفع جداً ولا يلبي متطلبات المعالجة في الزمن الحقيقي. يعزى هذا التأخر الكبير بشكل أساسي إلى سلسلة من عمليات الإدخال/الإخراج الشبكي (Network I/O) التي يجب أن تحدث لكل حدث: (1) طلب خدمة الاستعلام، (2) استهلاك الحدث من Kafka، (3) الاستعلام من Redis لجلب متجه المستخدم ، (4) تنفيذ بحث متجهي على Redis، (5) نشر النتيجة إلى Kafka. كل هذه العمليات، عند تنفيذها تحت ضغط 250 مستخدماً متزامناً على عنقود Dataproc بعقدة واحدة، أدت إلى تراكم الطلبات في قائمة الانتظار. إن النتيجة لا تعكس فشلاً في منطق Flink، بل تؤكد أن تطبيقات معالجة التدفق الحقيقية تتطلب بنية تحتية مخصصة ذات موارد كافية (عدة عقد TaskManager) لتوزيع هذا الحمل وتحقيق زمن استجابة منخفض. لذلك، يمكن تصنيف الأداء الحالي بأنه Stateful Streaming وليس Low-Latency Real-time.