ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

STEAM ANALYTICS: REAL-TIME TELEMETRY & CRISIS DETECTION

Group 22

20225473 - Nguyễn Trọng Phương Bách
20225494 - Vũ Minh Hiếu
20225501 - Bùi Nguyên Khải
20225523 - Trần Ngọc Quang
20225533 - Nguyễn Việt Tiến

ONE LOVE. ONE FUTURE.

The Problem Context

The Scale

- Steam Platform: 50,000+ games, 120M+ monthly active users.
- Data Velocity: Thousands of reviews and player status updates every second.

The Big Data Challenge (4 Vs)

- Volume: Massive datasets of historical and incoming reviews.
- Velocity: Real-time demands; waiting for daily batches is too slow.
- Variety: structured, semi-structured, time-series
- Verification: Need to handle strict API rate limits.

Core Issue

- Traditional metrics are reactive.
- Developers often miss "Review Bombing" attacks (coordinated negative campaigns) until the damage is done.

Project Goals & Innovation

Objective

- Build a production-grade, containerized real-time analytics pipeline.
- Goal: Reduce insight latency from 24 hours to <1 minute.

Key Innovation: Review Bomb Detection

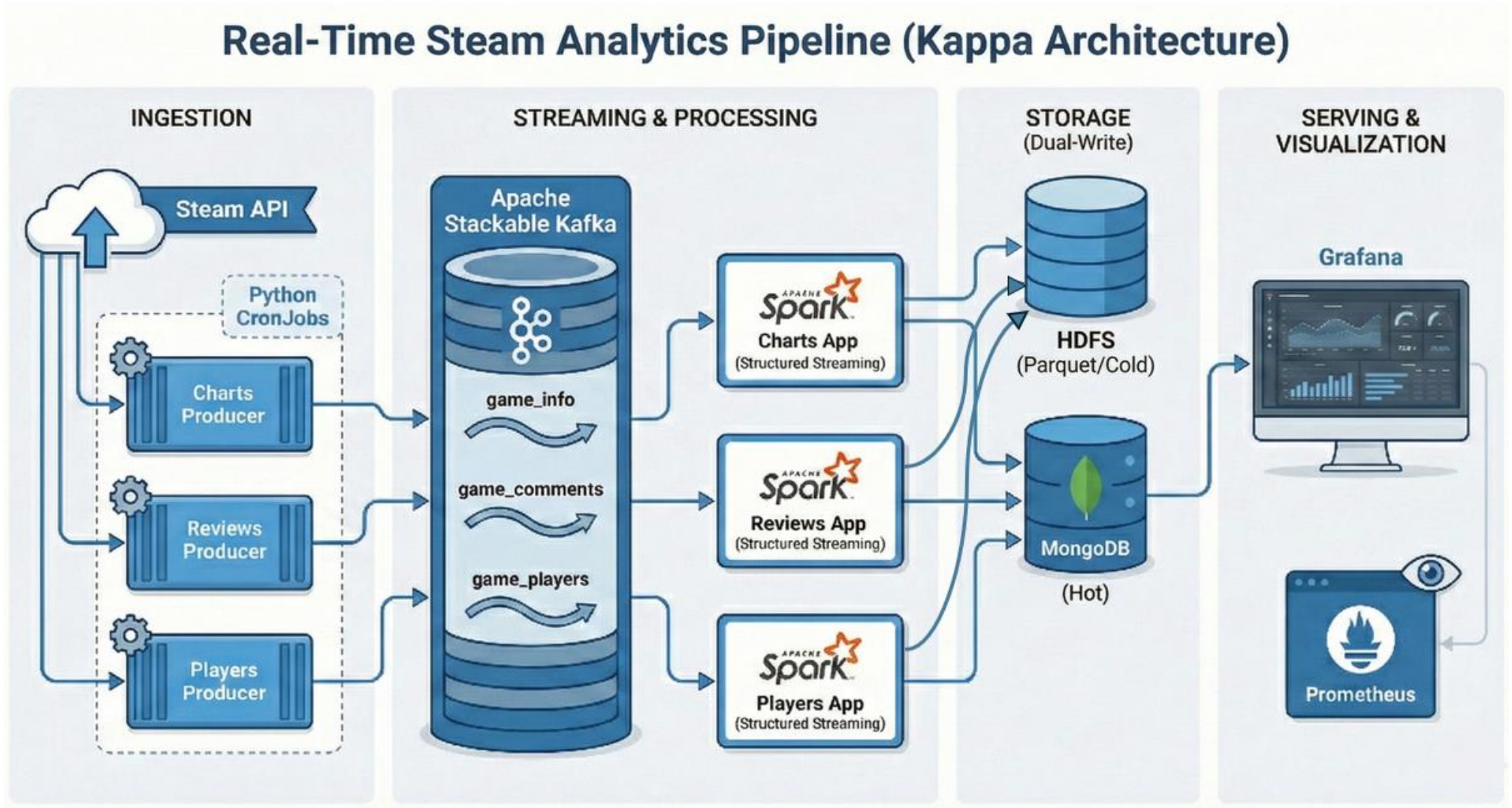
- Algorithm: Real-time flagging of "Panic Events".
- Trigger Formula:

$$\frac{R_{\text{negative}}}{R_{\text{total}}} > 0.8 \quad \text{AND} \quad R_{\text{total}} > 10$$

User Scenerio

- **Before:** Team wakes up to 1,000 negative reviews. Damage done.
- **After:** Dashboard Alert triggers in **30 seconds**. Immediate rollback.

System Architecture Overview



Technology Stack

Platform: Kubernetes (Docker Desktop).

Management: Stackable Data Platform

-**Core Infrastructure Components:**

- **Apache Zookeeper:** The "Brain" of the cluster.
  - Role: Distributed coordination, Leader Election for Kafka, and High Availability (HA) for HDFS NameNodes.
- **Stackable Operators:**
  - Role: Automates the complex lifecycle (deployment, scaling, upgrades) of Zookeeper, Kafka, and HDFS on Kubernetes.
- **Streaming Engine:** Spark 3.5.6 + Kafka 3.9.1.
- **Visualization:** Grafana + Prometheus + MongoDB Express.

Data Ingestion

- Data producer:
  - **Reviews & Info**: Python-based Kubernetes CronJob (producer_reviews.py, poducer_charts.py) that scrapes app details and user reviews.
  - **Player Counts**: Python-based Kubernetes CronJob (producer_players.py) running every 5 minutes to fetch real-time player statistics.
- Kafka Cluster Structure:
  - **Deployment**: Hosted on Kubernetes using the Stackable Operator.
  - **Configuration**:
    - **Single Broker** setup optimized for development/testing (1 replica).
    - **Retention Policy**: 7 days (168 hours) to support Kappa architecture replayability.
    - **Partitions**: 3 Partitions per topic.
    - **Resources**: Limited to 1 CPU and 2Gi Memory per broker.
    - **Zookeeper**: Integrated for cluster coordination.
- **Transport Encryption**: All traffic between producers and the Kafka cluster is encrypted using **SSL/TLS**.

Data Ingestion

- Kafka message structure:
  - Player count:

```
player_record = {
    "appid": int(appid),
    "player_count": p_count,
    "timestamp": datetime.utcnow().isoformat()
}
```

  - Game info:

```
return {
    "appid": int(appid),
    "name": item.get("name") or data.get("name"),
    "primary_genre": genre_tag,
    "type": data.get("type"),
    "release_date": data.get("release_date", {}).get("date"),
    "is_free": data.get("is_free"),
    "short_description": clean_html(data.get("short_description")),
    "developers": data.get("developers") or [],
    "publishers": data.get("publishers") or [],
    "genres": [g.get("description") for g in (data.get("genres") or [])],
    "price_overview": data.get("price_overview") or {},
    "categories": [c.get("description") for c in (data.get("categories") or [])],
    "metacritic": data.get("metacritic", {}).get("score"),
    "recommendations": data.get("recommendations", {}).get("total"),
    "achievements_count": data.get("achievements", {}).get("total", 0),
    "timestamp_scraped": datetime.utcnow().isoformat()
}
```

Data Ingestion

- Kafka message structure:
  - Game review:

```
yield {
    "app_id": app_id,
    "review_id": r.get("recommendationid"),
    "author_steamid": r.get("author", {}).get("steamid"),
    "playtime_at_review": r.get("author", {}).get("playtime_at_review"),
    "playtime_forever": r.get("author", {}).get("playtime_forever"),
    "language": r.get("language"),
    "voted_up": r.get("voted_up"),
    "votes_up": r.get("votes_up"),
    "weighted_vote_score": r.get("weighted_vote_score"),
    "timestamp_created": r.get("timestamp_created"),
    "review_text": clean_html(r.get("review")),
    "scraped_at": datetime.utcnow().isoformat()
}
```

Stream Processing Logic

- **Charts App**
  - **Genre Explosion:** Unnests arrays (e.g., **`[Action, RPG]`**) via **`explode`** for accurate category counts.
- **Reviews App**
  - **Windowing:** Uses **1-hour tumbling windows** with a **10-minute watermark** to manage late data .
  - **Metrics:** Aggregates `total_reviews` and avg quality to track sentiment .
- **Players App**
  - **Granularity:** Uses **10-minute windows** to capture rapid activity fluctuations.
- **Resilience Features**
  - **Schema Validation:** Strict **`StructType`** parses invalid JSON as **`null`** to prevent crashes.
  - **Watermarking:** Discards records exceeding the 5-minute threshold to prevent memory growth.
  - **Checkpointing:** Stores query state in dedicated HDFS directories (e.g., **`/checkpoints/reviews_cold`**) to support restartability.

Storage Strategy

- **The Dual-Write Approach:**
  - **Cold Storage – Bronze Layer (Raw data): Apache HDFS**
    - **Purpose:** Archival, long-term retention
    - **Format:** Parquet (Columnar storage).
    - **Benefit:** 10:1 Compression ratio compared to raw JSON.
    - **Path:** `/archive/reviews`, `/archive/charts`, `/archive/players`.
    - **Use Case**: DL/ML models training or debugging pipeline logic.
  - **Hot Storage – Gold Layer (Aggregated data): MongoDB**
    - **Purpose:** Business-level aggregates ready for consumption (operational analytics, dashboards, web app API)
    - **Format:** Optimized Documents with Arrays/Counters.
    - **Optimization:** TTL Indexes (30 days) to prevent storage bloat and maintain query speed.
    - **Use case:** Powering the Real-Time Grafana Dashboard.
  - **Note:** We skip a persistent "Silver" layer to minimize latency (Stream-to-Gold).

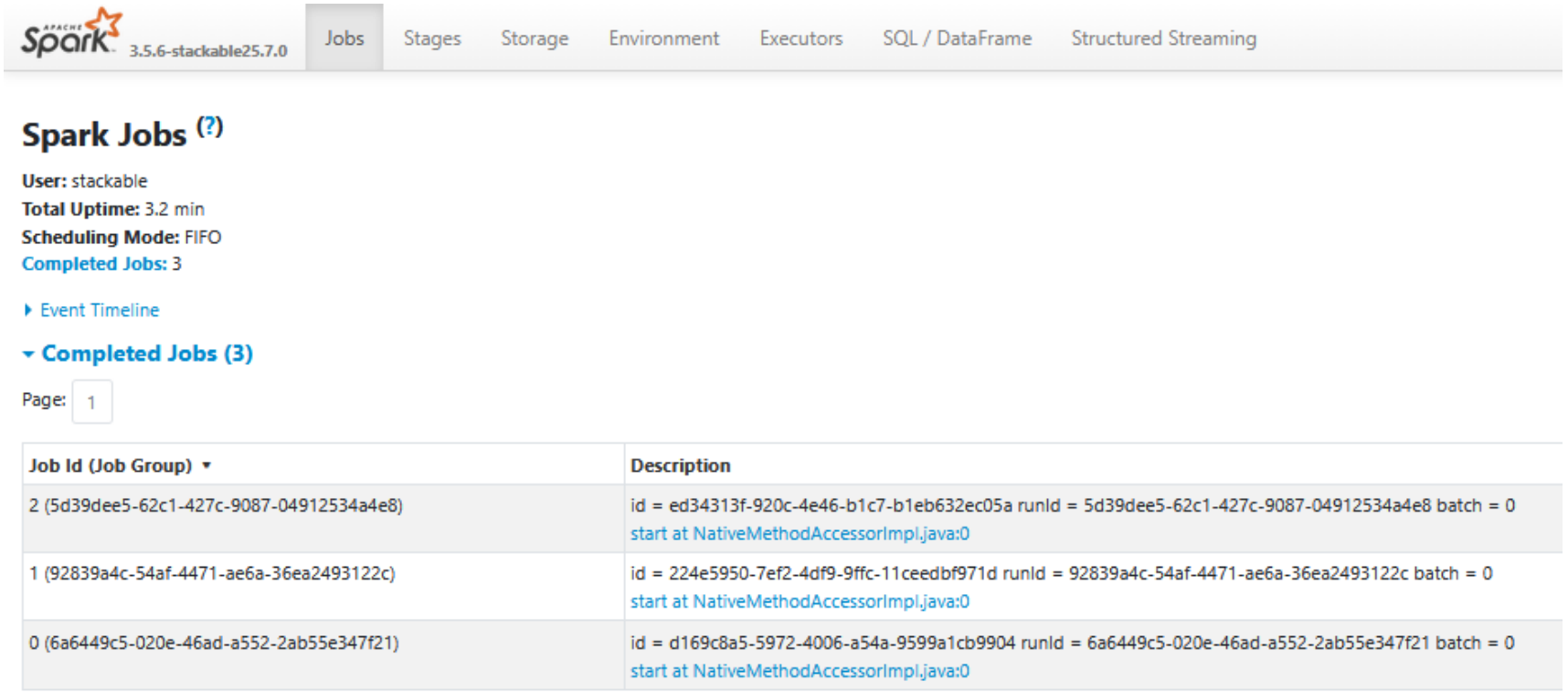
Storage Strategy

HDFS Schema (Parquet)

- **Reviews**
  - app_id (String)
  - review_id (String)
  - recommended (Boolean)
  - votes_up (Integer)
  - weighted_vote_score (Float)
  - timestamp (Timestamp) :: Event Time
- **Charts**
  - appid (Integer)
  - name (String)
  - type (String) :: e.g., "game", "dlc"
  - genres (Array[String]) :: e.g. ["Action", "FPS"]
  - timestamp (Timestamp) :: Scrape Time
- **Players**
  - appid (Integer)
  - player_count (Integer)
  - timestamp (Timestamp) :: Observation Time

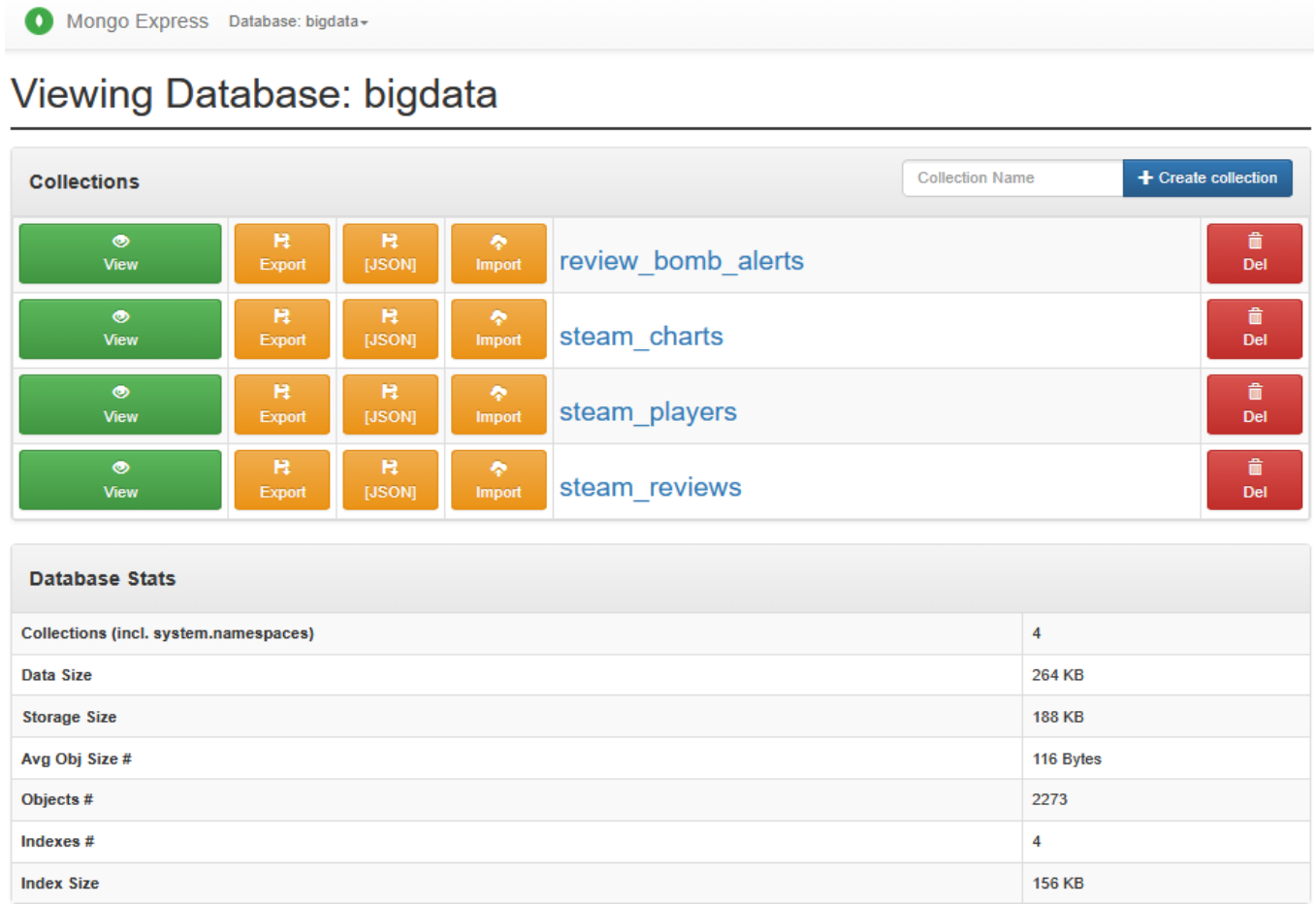
Visualization

- **Spark UI:** A comprehensive interface to monitor and analyze the execution of Spark applications and offers insights into the status and resource consumption of Spark cluster.



Visualization

- **Mongo Express:** A web-based admin interface for browsing MongoDB collections and running ad-hoc queries.

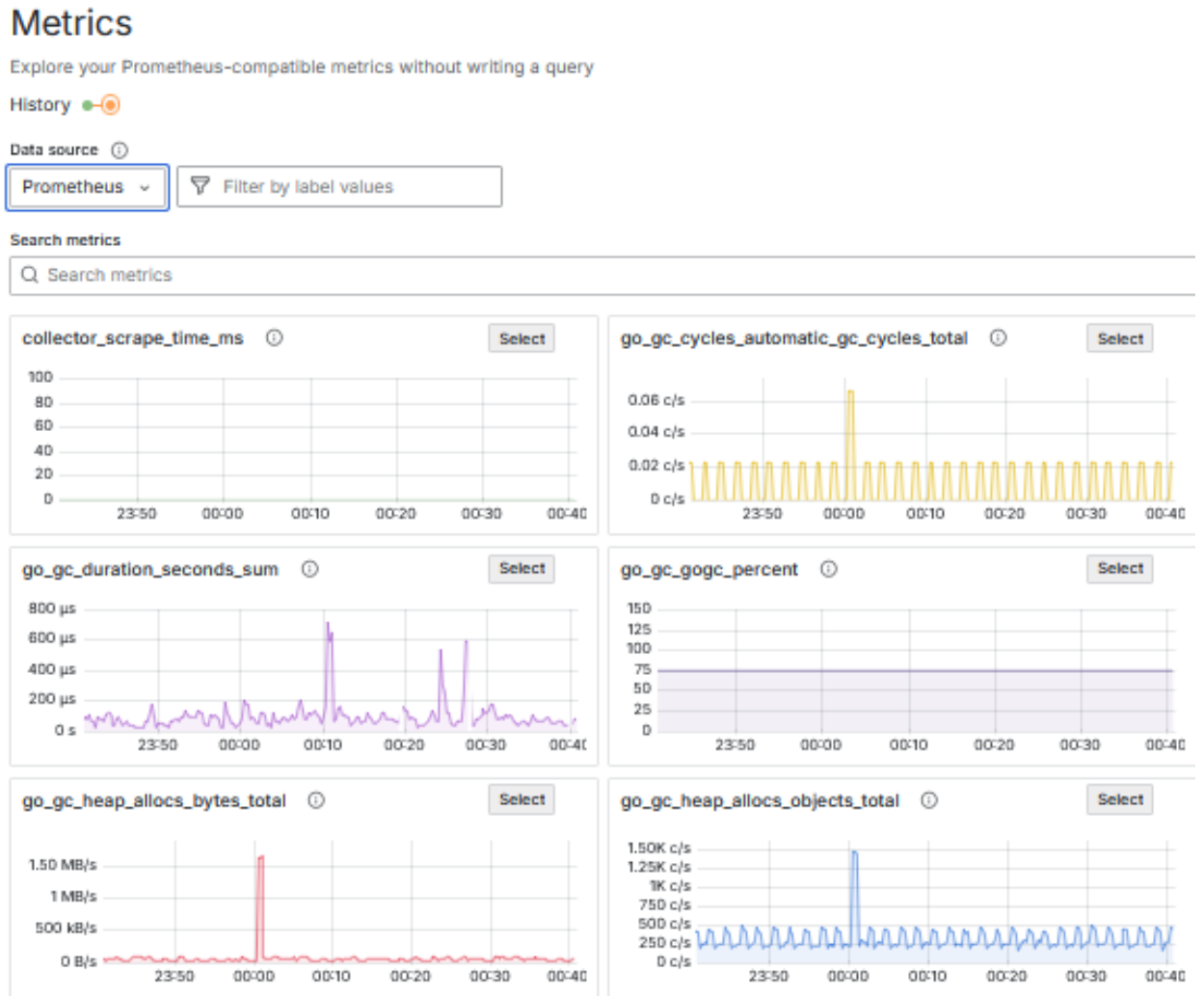


Visualization

- **Grafana:** An open-source analytics and monitoring platform that let users query, visualize, and alert on metrics from various data sources.
- Usage:
  - Monitor system metrics of databases (MongoDB and Prometheus).
  - Display key features of collected Steam data.

Visualization

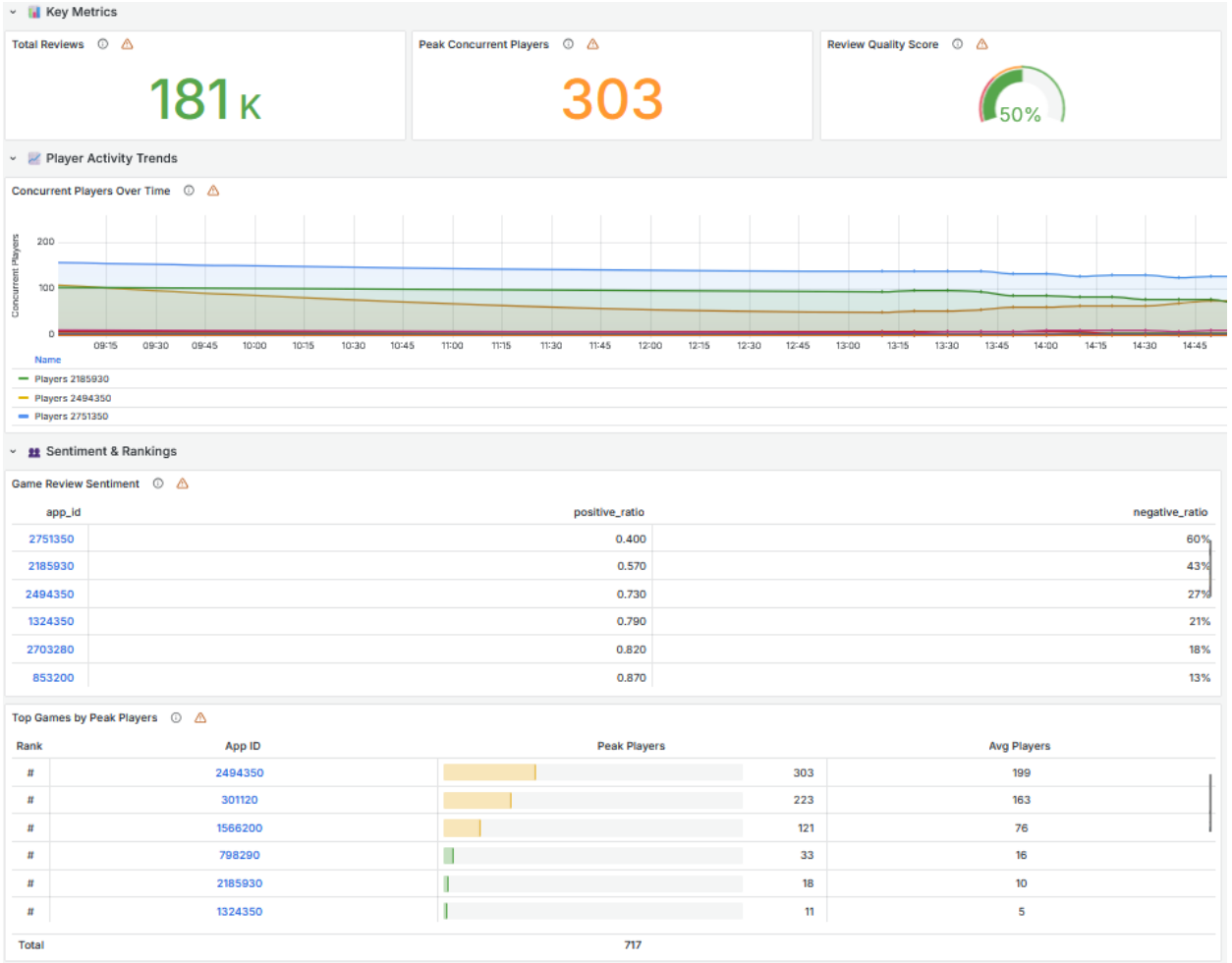
- Monitor system metrics of databases:
  - Prometheus collects and stores time-series metrics (such as CPU/memory usage, request counts) from systems and applications.
  - Information is shown in Grafana's Drilldown section.



Visualization

- Display key features of collected Steam data:
  - **Review Bomb alert:** Green/red status indicator of abnormal surge in negative reviews.
  - **Top games**: Show top games by number of players across Steam.
  - **Performance check:** Real-time concurrent player counts.

  … and more.



Lesson Learned

1. The "TLS Nightmare" (Security Mismatch)

- Context: Stackable enforces TLS, but Spark Java clients expect JKS/PKCS12.
- Fix: Implemented Kubernetes Init Containers to run keytool conversion before the main application starts.

2. The "Small Files" Problem (HDFS)

- Context: Real-time streaming generates thousands of tiny KB-sized files, threatening NameNode stability.
- Fix: Increased HDFS write trigger to 1 minute (Micro-batching) to reduce file creation frequency by 60x compared to continuous processing.

3. Running a Data Center on a Laptop (Resources)

- Context: Running Zookeeper, Kafka, HDFS, Spark, and Mongo on Docker Desktop (8GB RAM limit).
- Fix: Aggressive tuning:
  - Reduced Zookeeper/NameNodes to 512MB.
  - Allocated 2GB to Kafka (for buffering).
  - Running single replicas instead of full HA for dev.

Business Impact & Conclusion

- **Speed:** Insight generation reduced from 24+ hours to 30 seconds.
- **Efficiency:** Replaced manual analyst monitoring with automated alerts.
- **Scalability:** Zero-downtime scaling via Kubernetes replicas.

- **Path to Production Scale**
  - Storage: Upgrade HDFS to Delta Lake for ACID transactions and automatic small-file compaction.
  - Infrastructure: Migrate to Managed K8s (EKS/AKS) to scale workers dynamically.
  - Governance: Add Confluent Schema Registry to enforce strict API contracts.
  - Dashboard: Hype-to-Disappointment Index (predict refund waves during a launch)

Conclusion: Kappa Architecture is a practical, robust, and efficient solution for handling the velocity and volume of modern gaming telemetry.

HUST

hust.edu.vn fb.com/dhbkhn

THANK YOU !