

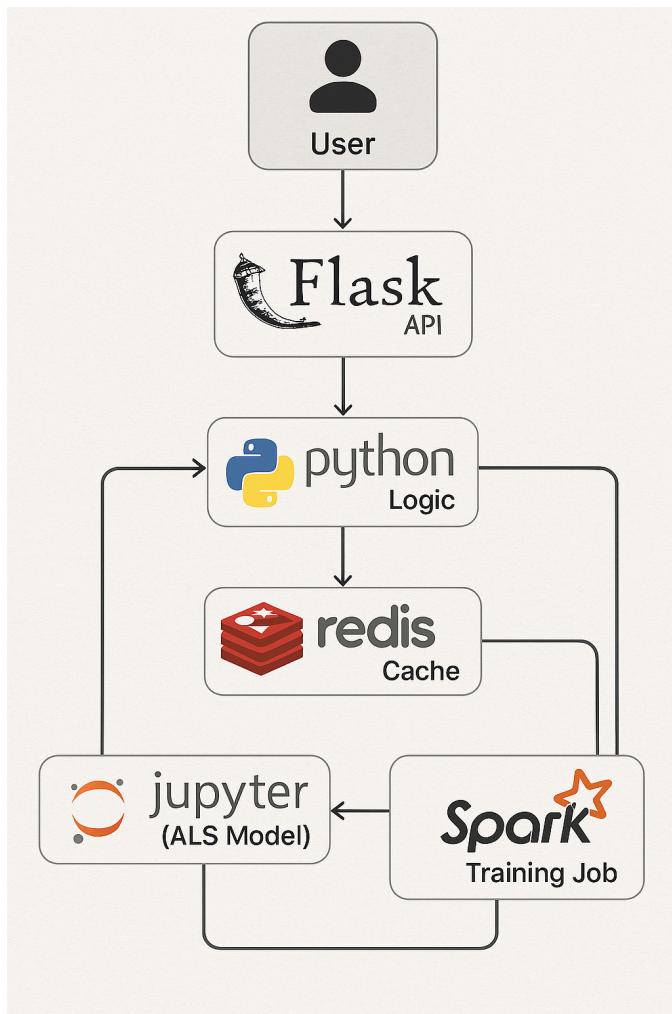
Netflix- Redis-Cache-Recommendation Engine

A full-stack, end-to-end ML pipeline simulating a real-world recommendation system for a streaming service. This project includes data ingestion, model training with Spark, REST API serving via Flask, Redis caching, Prometheus + Grafana monitoring, and containerized orchestration using Docker Compose.

🔧 Technologies Used

- Docker + Docker Compose
- Apache Spark (PySpark)
- Redis
- Flask
- Prometheus & Grafana
- Jupiter Notebook (for exploration)

🚀 System Architecture



Project Structure

```
.  
├── docker-compose.yml  
├── prometheus.yml  
├── flask-api/  
│   ├── app.py  
│   ├── Dockerfile  
│   └── requirements.txt  
├── jupyter/  
│   └── train_model.py  
└── data/  
    └── synthetic_streaming_data.csv  
grafana/ (provisioning dashboards)
```

Docker Compose Setup

```
docker-compose up --build
```

Access:

- Flask API: <http://localhost:5000>
 - Prometheus: <http://localhost:9090>
 - Grafana: <http://localhost:3000>
 - Jupyter Notebook: <http://localhost:8888>
-

Prometheus Configuration

```
global:  
  scrape_interval: 15s  
  evaluation_interval: 15s  
  scrape_timeout: 10s  
  
scrape_configs:  
  - job_name: 'prometheus'  
    static_configs:  
      - targets: ['prometheus:9090']  
  
  - job_name: 'node-exporter'  
    static_configs:  
      - targets: ['metrics-server:9100']  
  
  - job_name: 'flask-api'  
    static_configs:  
      - targets: ['flask-api:5000']  
  
  - job_name: 'jupyter-notebook'  
    static_configs:  
      - targets: ['host.docker.internal:8001']
```

Flask API Endpoints

- GET /health - Health check
- POST /watched - Simulates a user watching a film

```
{  
    "user_id": "U1001",  
    "content_id": "C1010"  
}
```

- GET /recommend/<user_id> - Returns recommendation
- GET /metrics - Prometheus metrics

Monitoring

- Prometheus scrapes metrics from Flask and Jupyter
 - Grafana dashboards show:
 - Cache hits & misses
 - Fallback usage
 - Model training runtime
-

ML Training Pipeline

Location: train_model.py

- Loads streaming data
- Encodes user_id and content_id
- Trains ALS (collaborative filtering) model
- Caches top recommendation for each user into Redis

Sample Output:

```
✓ Cached for U1146: C384  
✓ Cached for U1198: C892  
...
```

Screenshots

Recommendation Output:

```

rahul@Rahul MINGW64 ~/Desktop/Data Engineering/project-netflix (master)
$ curl -X POST http://localhost:5000/watched \
  -H "Content-Type: application/json" \
  -d '{"user_id": "U1001", "content_id": "C1010"}'
{"predicted_next":"C1045","user_id":"U1001","watched":"C1010"}

rahul@Rahul MINGW64 ~/Desktop/Data Engineering/project-netflix (master)
$ 

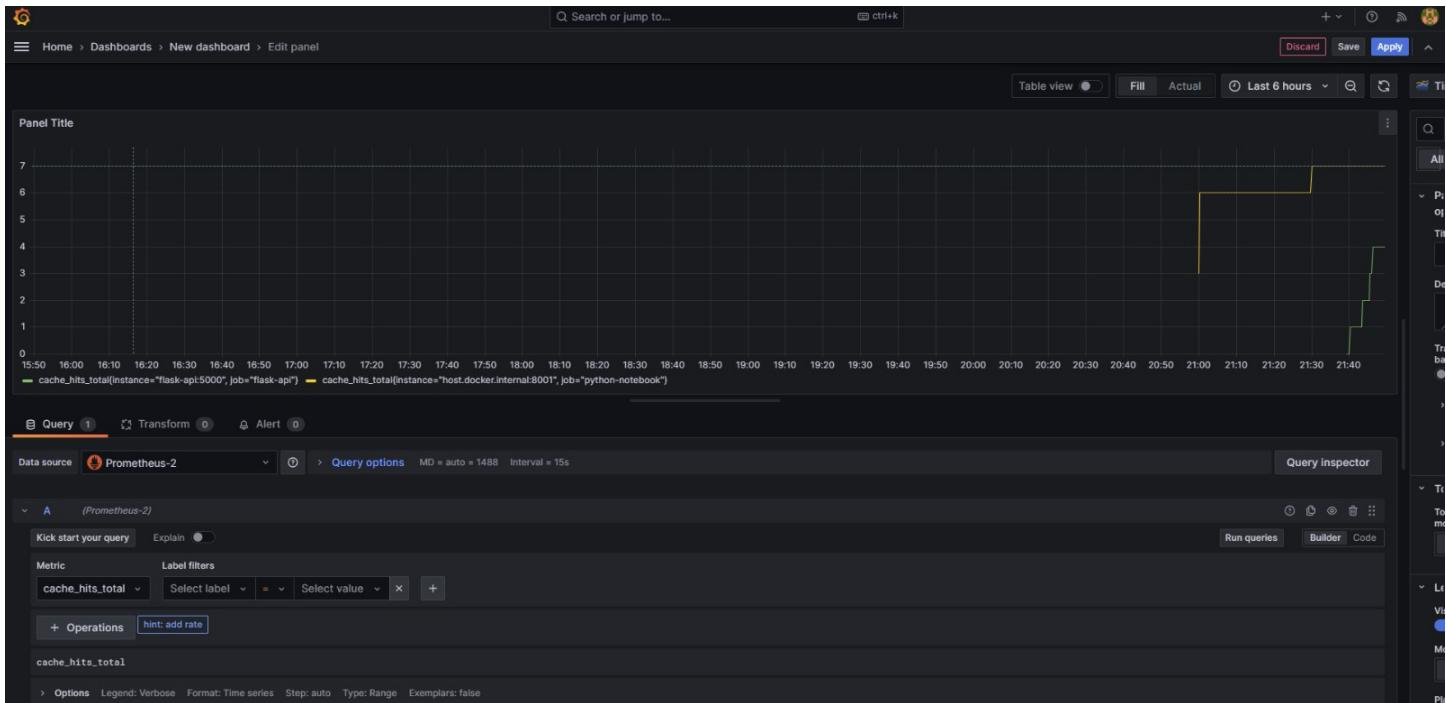
```

```

rahul@Rahul MINGW64 ~/Desktop/Data Engineering/project-netflix (master)
$ curl http://localhost:5000/recommend/U1001
{"cache_hit":true,"recommendation":"C1045","user_id":"U1001"}

```

Grafana Dashboard:



API with Prometheus Metrics:

Prometheus Alerts Graph Status Help

Targets

All scrape pools ▾ All Unhealthy Collapse All Filter by endpoint or labels Unknown Unhealthy Healthy

flask-api (1/1 up)		node-exporter (1/1 up)		prometheus (1/1 up)		python-notebook (1/1 up)	
Endpoint	State	Labels		Last Scrape	Scrape Duration	Error	
http://flask-api:5000/metrics	UP	instance="flask-api:5000" job="flask-api"		46.667s ago	3.550ms		
http://metrics-server:9100/metrics	UP	instance="metrics-server:9100" job="node-exporter"		46.895s ago	60.251ms		
http://prometheus:9090/metrics	UP	instance="prometheus:9090" job="prometheus"		44.361s ago	17.656ms		
http://host.docker.internal:8001/metrics	UP	instance="host.docker.internal:8001" job="python-notebook"		40.315s ago	10.085ms		

✓ TODO Next

- Add Kafka ingestion
- Replace fallback logic with real-time inference
- Federated learning for edge recommendations

✋ Maintainer

Built by [Rahul Raj](#) - striving for ML-powered personalized streaming.

📜 License

MIT License

🎬 Netflix-Style Recommendation System Demo

✍️ Overview

This project showcases a real-time, containerized recommendation pipeline using Docker Compose. It integrates services like Spark, Flask API, Redis, Prometheus, and Grafana to simulate a Netflix-style recommendation system.

✓ Step 1: Show the System is Running

🎙️ Narration:

"I've containerized and automated the entire Netflix-style recommendation pipeline using Docker Compose. Everything you see here is real-time and monitored."

🔧 Command:

```
bash
CopyEdit
docker ps
```

🔍 Expected Output:

CONTAINER ID	IMAGE	COMMAND	STATUS	PORTS
abc123	spark	"/bin/bash"	Up 5 minutes	...
def456	jupyter	"start-notebook.sh"	Up 5 minutes	...
ghi789	flask-api	"python app.py"	Up 5 minutes	5000/tcp
JKL1012	redis	"docker-entrypoint.sh"	Up 5 minutes	6379/tcp
mno345	prometheus	"/bin/prometheus"	Up 5 minutes	9090/tcp
pqr678	grafana	"/run.sh"	Up 5 minutes	3000/tcp

🖼️ Screenshot Placeholder:

Insert a screenshot of the terminal displaying the docker ps output.

✓ Step 2: Simulate a User Watching a Movie

🎙️ Narration:

"Let's say user U999 watches C1010, which is, say, Stranger Things. The system automatically predicts and caches the next likely show."

🔧 Command:

```
bash
CopyEdit
curl -X POST http://localhost:5000/watched \
-H "Content-Type: application/json" \
-d '{"user_id": "U999", "content_id": "C1010"}'
```

⌚ Expected Response:

```
json
CopyEdit
{
  "user_id": "U999",
  "watched": "C1010",
  "predicted_next": "C1045"
}
```

🖼 Screenshot Placeholder:

Insert a screenshot of the terminal displaying the curl response.

✓ Step 3: Retrieve Cached Recommendation

🎤 Narration:

“Now, the user comes back to the app and asks: what should I watch next? We instantly return the prediction from Redis.”

🔧 Command:

```
bash
CopyEdit
curl http://localhost:5000/recommend/U999
```

⌚ Expected Response:

```
json
CopyEdit
{
  "user_id": "U999",
  "recommendation": "C1045",
  "cache_hit": true
}
```

🖼 Screenshot Placeholder:

Insert a screenshot of the terminal displaying the curl response.

✓ Step 4: Handle New User Scenario

Narration:

“If the user is new and we don’t have predictions yet, the system handles it gracefully with a fallback.”

Command:

```
bash
CopyEdit
curl http://localhost:5000/recommend/U1234
```

Expected Response:

```
json
CopyEdit
{
  "user_id": "U1234",
  "recommendation": "C000",
  "cache_hit": false,
  "fallback_used": true
}
```

Screenshot Placeholder:

Insert a screenshot of the terminal displaying the curl response.

Step 5: Monitor System Metrics with Prometheus

Narration:

“We also expose Prometheus metrics for system health and activity.”

Access:

Navigate to <http://localhost:9090> in your browser.

Metrics to Observe:

- cache_hits
- cache_misses
- fallbacks_used

Screenshot Placeholder:

Insert a screenshot of the Prometheus dashboard displaying the relevant metrics.

Step 6: Visualize Data with Grafana

Narration:

“Here’s the Grafana dashboard tracking live recommendations, cache efficiency, fallback trends, and more.”

Access:

Navigate to <http://localhost:3000> in your browser.

Login Credentials:

- **Username:** admin
- **Password:** adminReddit+2Netflix TechBlog+2Medium+2Medium+12doc.ic.ac.uk+12Learn R, Python & Data Science Online+12

Screenshot Placeholder:

Insert a screenshot of the Grafana dashboard displaying the relevant panels.

Bonus Features

- Develop a simple HTML/Javascript frontend or CLI to simulate the watch and recommend flow.
- Display Spark job logs within JupyterLab or via Docker logs.
- Highlight the pipeline's extensibility for federated learning, edge caching, or A/B testing.[AWS in Plain English](#)