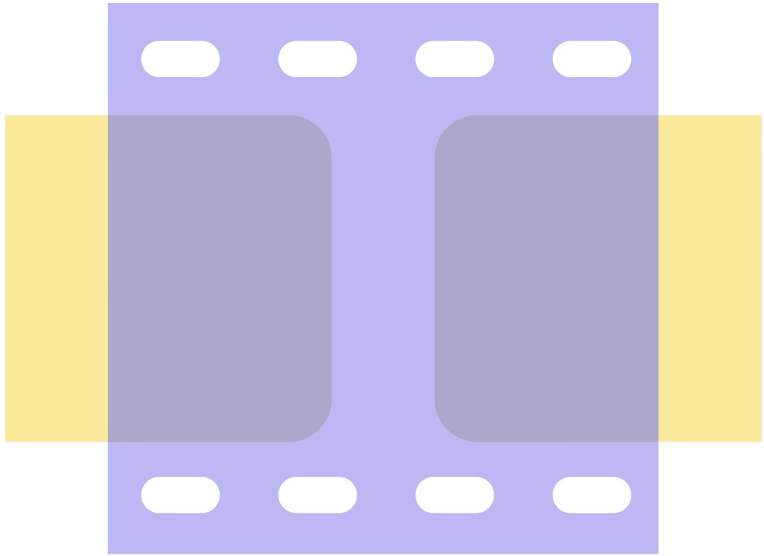
UC San Diego

Cinescope:

Scalable Movie Genre Search and Analysis



Group #18

Joel Polizzi, Dongting Cai, Xuanwen Hua

March 18, 2025

Introduction

Project Goals:

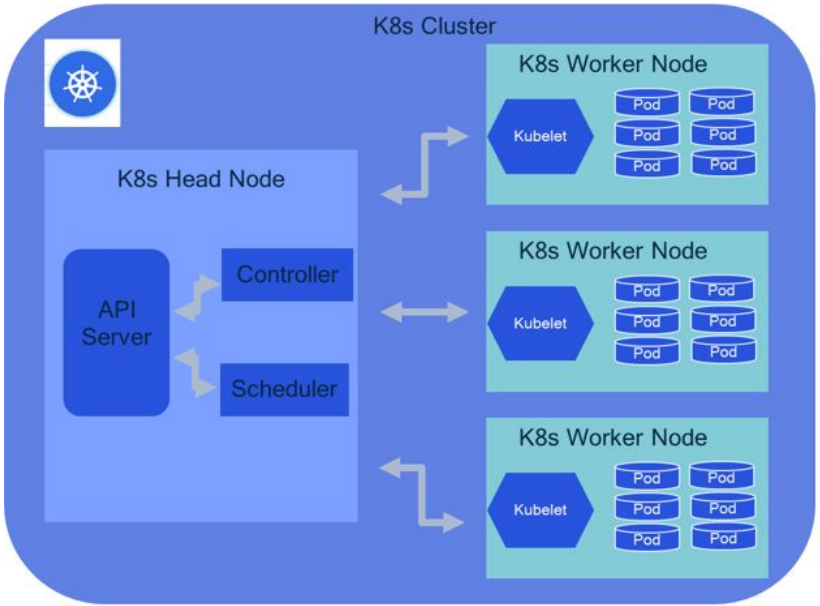
- Create a developer to production pipeline on a platform that allows for scalability
- Use up-to-date TMDB (the movie database) datasets
- Build a relational Genre search tool with PostgreSQL
- Use Neo4J to query based on Genre to analyze graph structures in the data
- Integrate Redis as a database cache
- Build a flask application that is publicly available (cinescope.nrp-nautilus.io)

Infrastructure

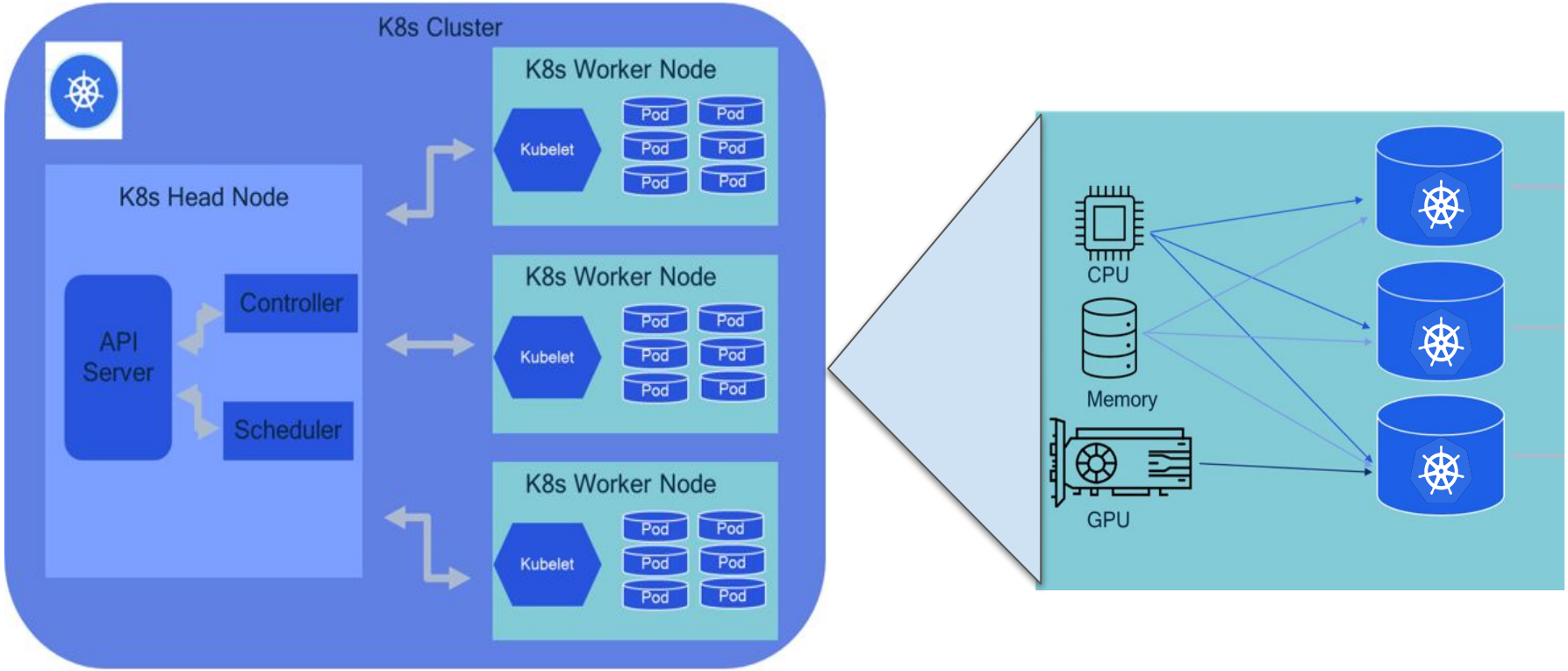
- High throughput networking on CENIC (10gbps to ~400gbps)
- Microservice architecture
- Easy to customize for our application
- Containers allow for application replication
- Storage and compute resources are readily available

Infrastructure

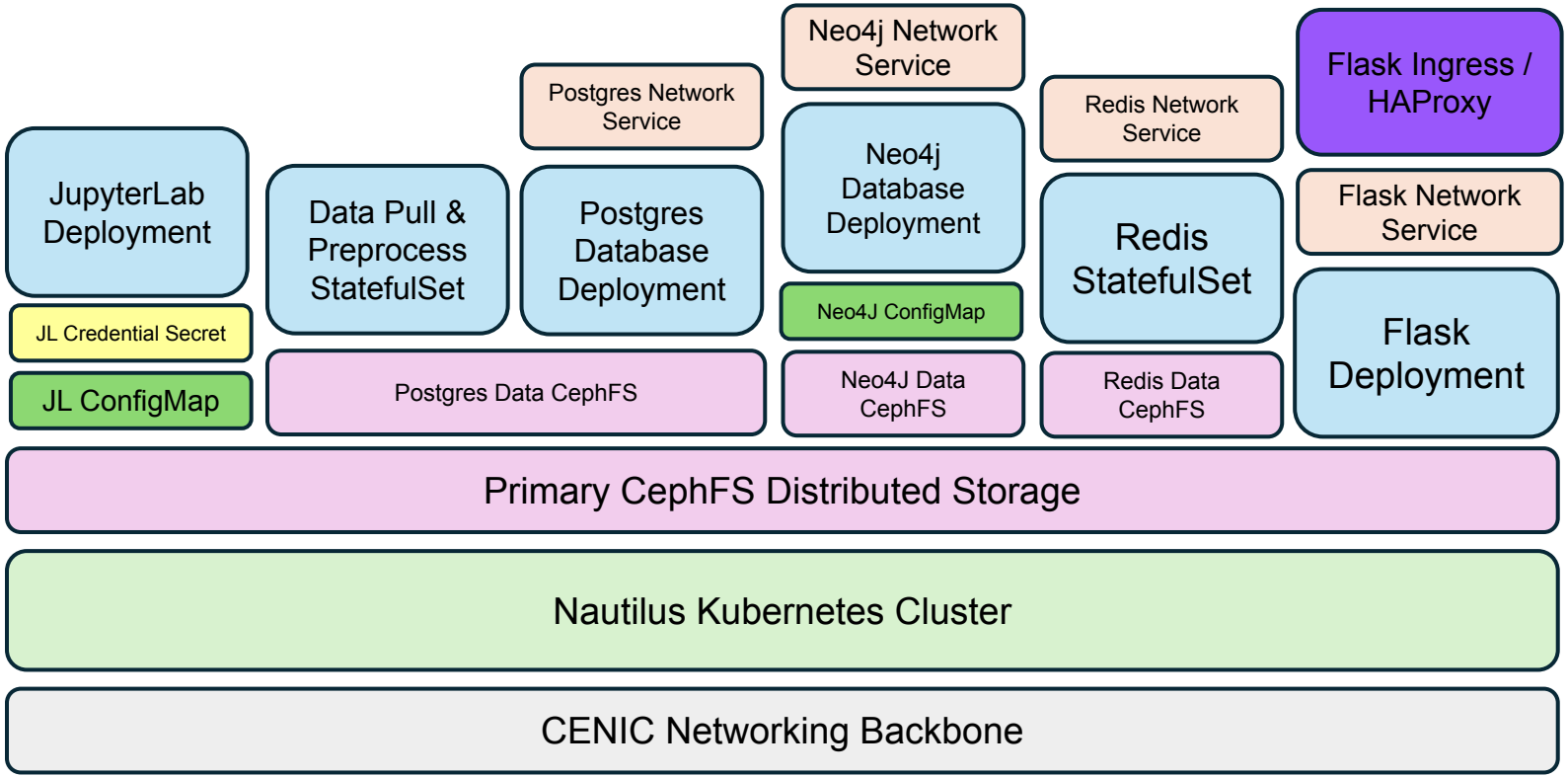
- High throughput networking on CENIC (10gbps to ~400gbps)
- Microservice architecture
- Easy to customize for our application
- Containers allow for application replication
- Storage and compute resources are readily available



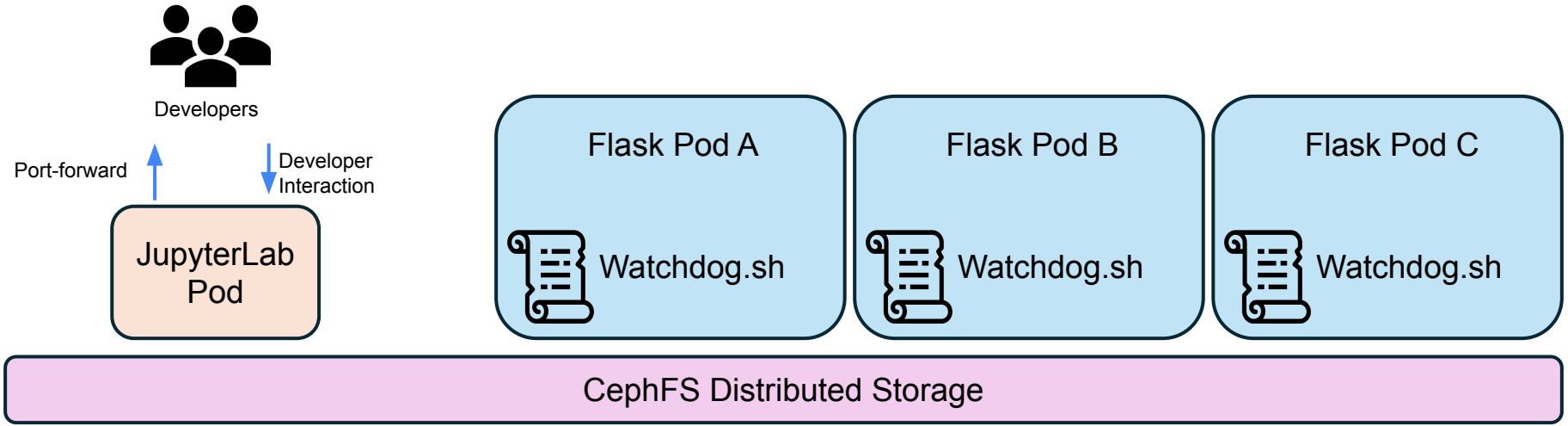
Infrastructure



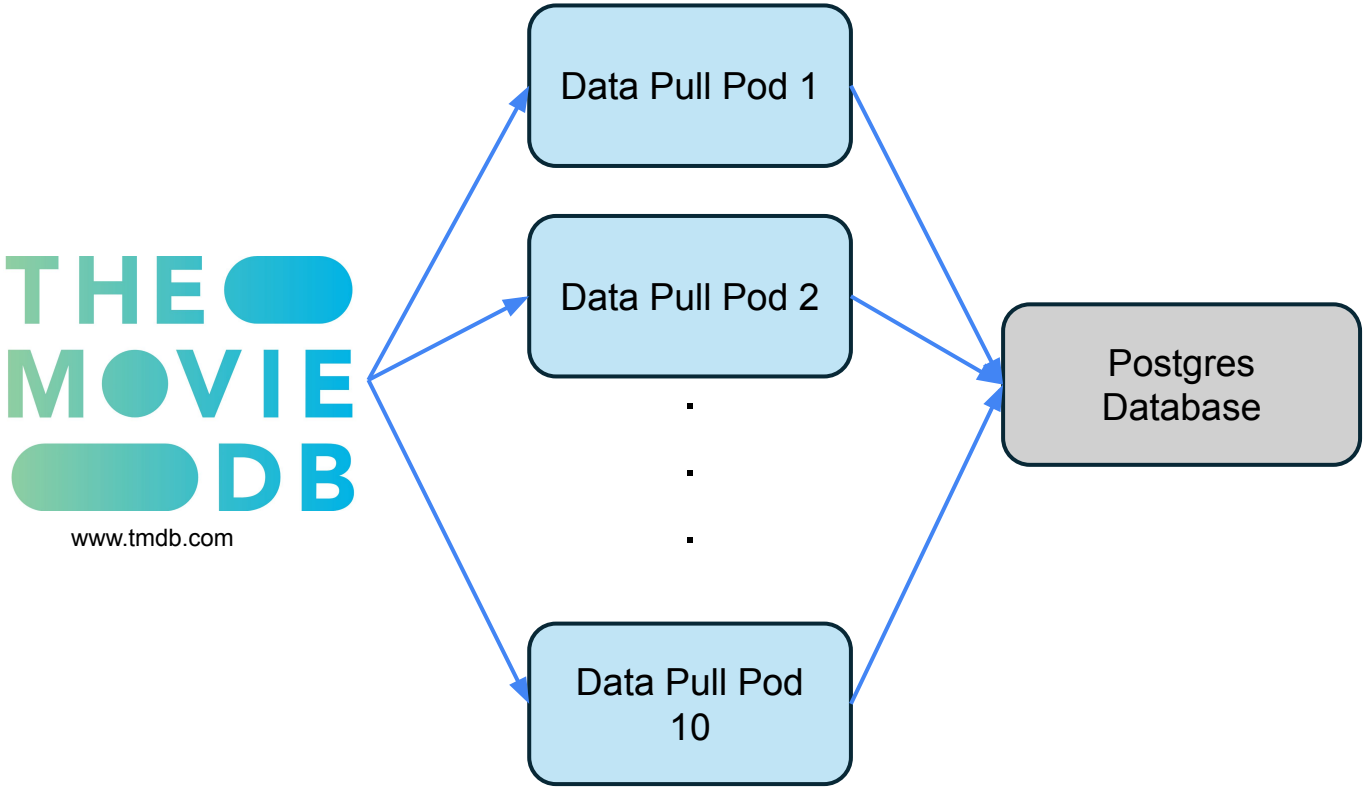
Cloud-Native Stack



From Development to Production

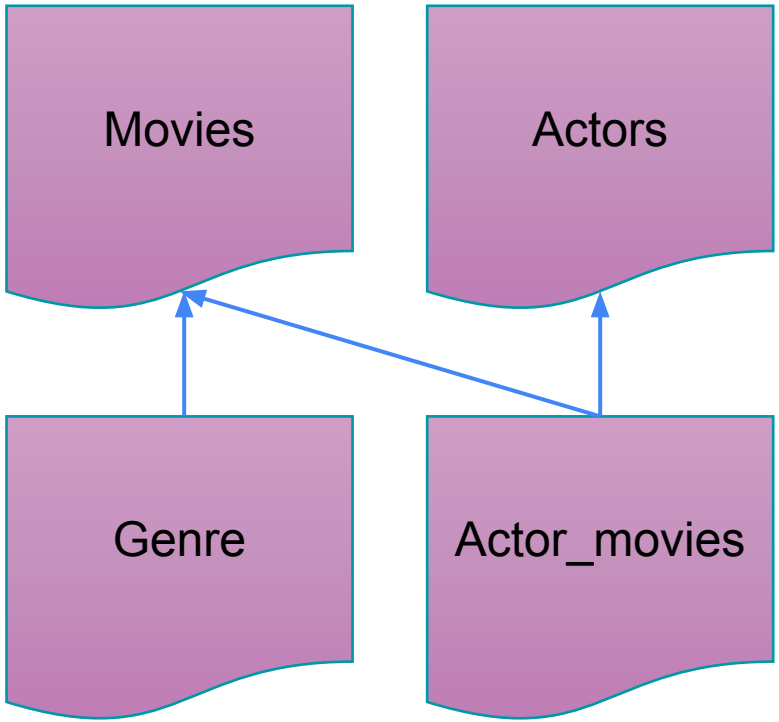


Data Collection



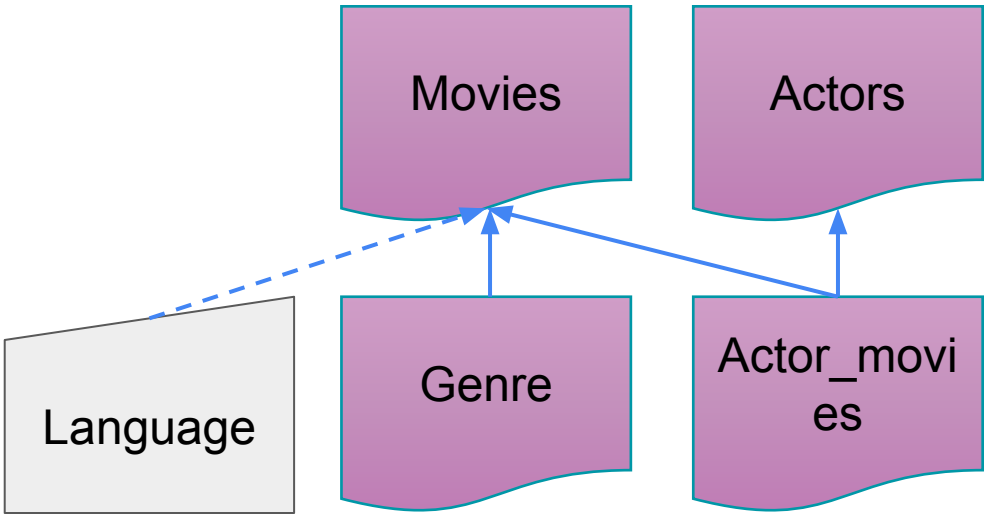
SQL Table Schema

- Movies table is the primary table referenced in the application.
- Actors table Contains an ID of an actor (or crew member) and associates the ID with their Name.
- Actor_movies bridges the Actors dataset and the Movies dataset.
- Genre table is used in conjunction with the Movies table



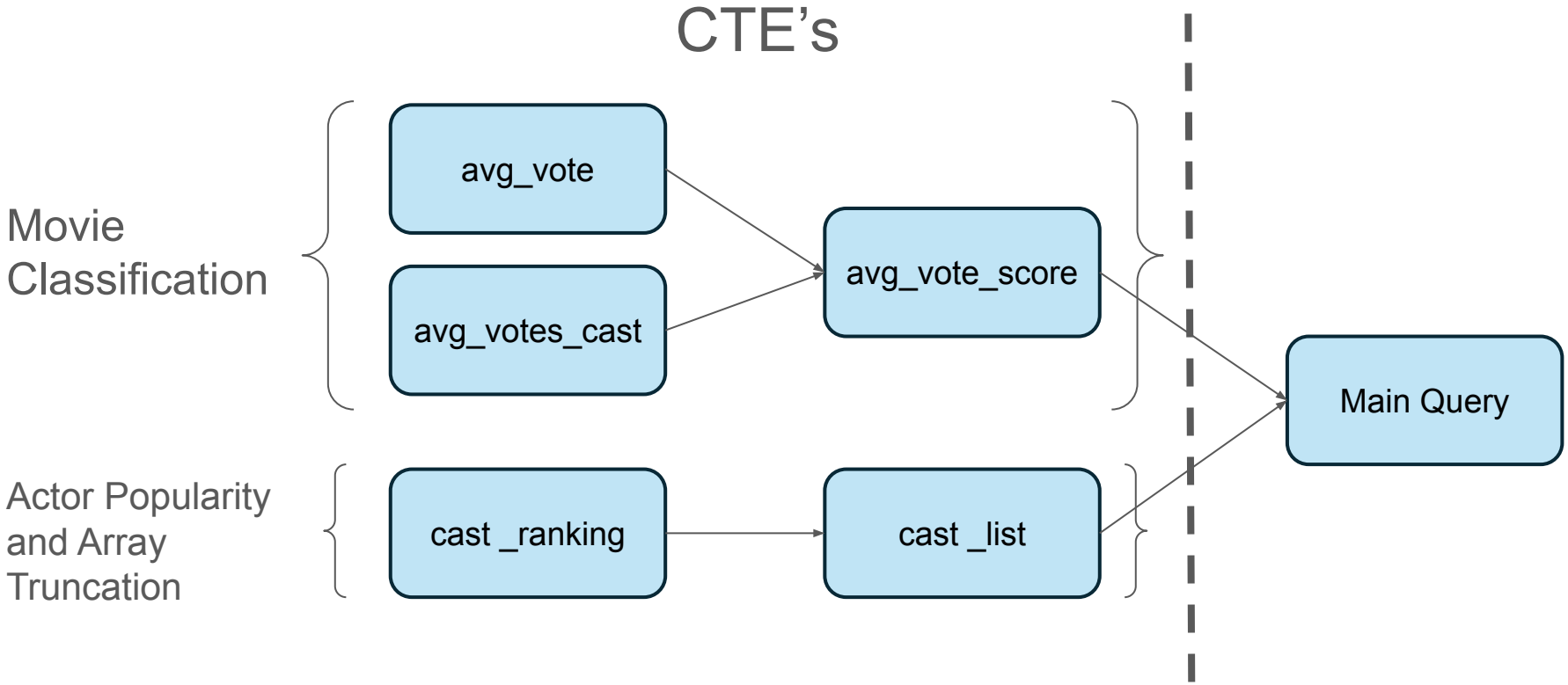
SQL Table Schema + Language

- Movies table is the primary table referenced in the application.
- Actors table Contains an ID of an actor (or crew member) and associates the ID with their Name.
- Actor_movies bridges the Actors dataset and the Movies dataset.
- Genre table is used in conjunction with the Movies table
- Language is introduced as a dictionary generated from querying the



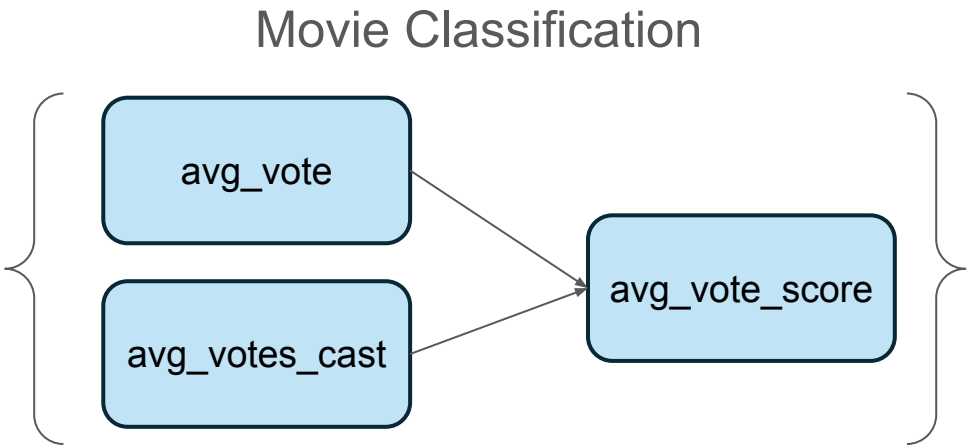
Pseudo-Star Schema

SQL Query Flow

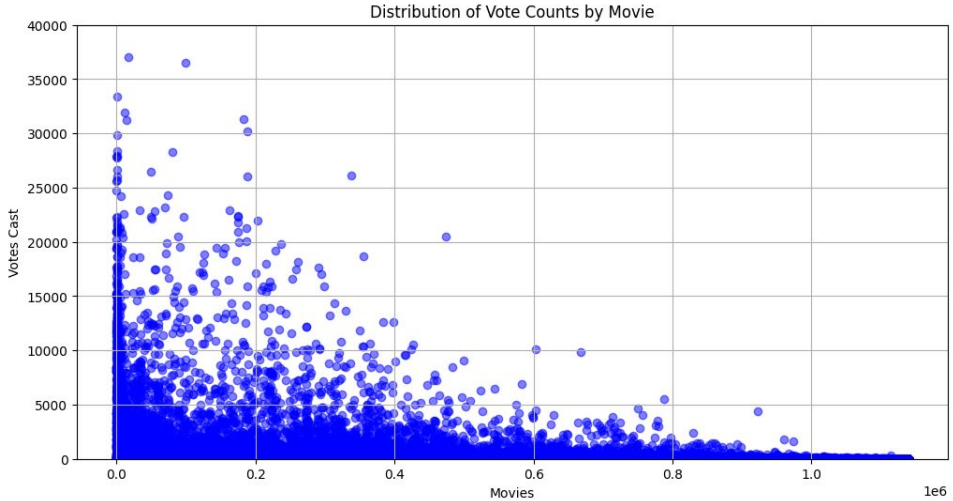


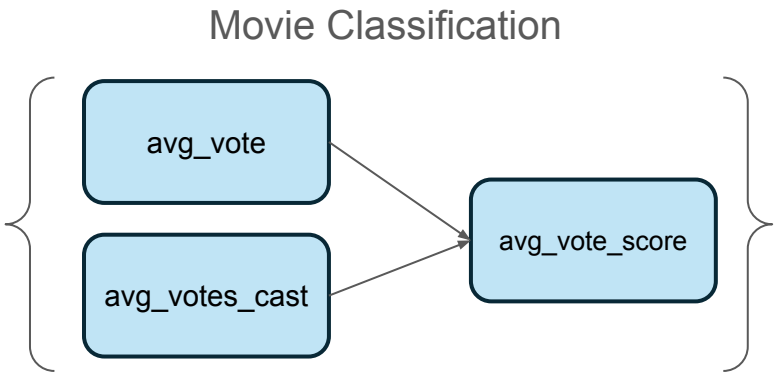
SQL Classification

- Average all vote averages
- Find the average of all votes cast
- Join the avg_vote CTE and the avg_votes_cast CTE into the avg_vote_score to design a classifier



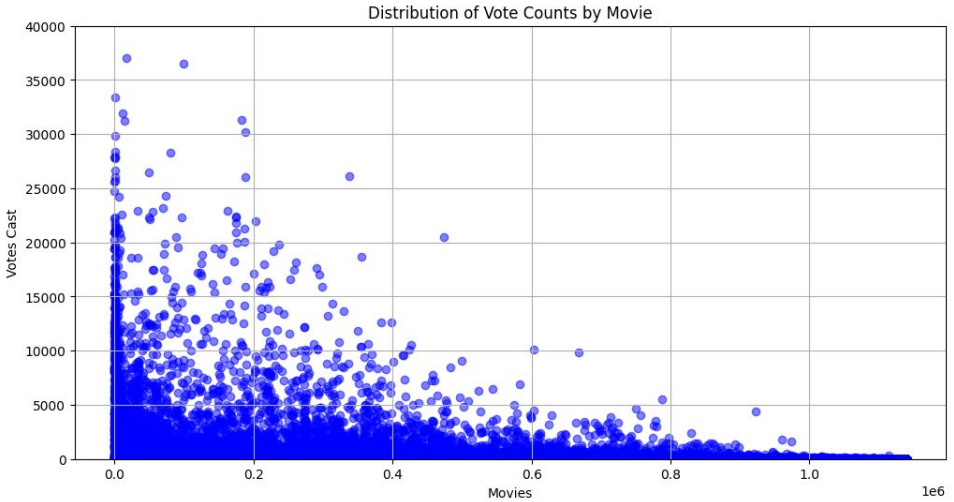
SQL Classification

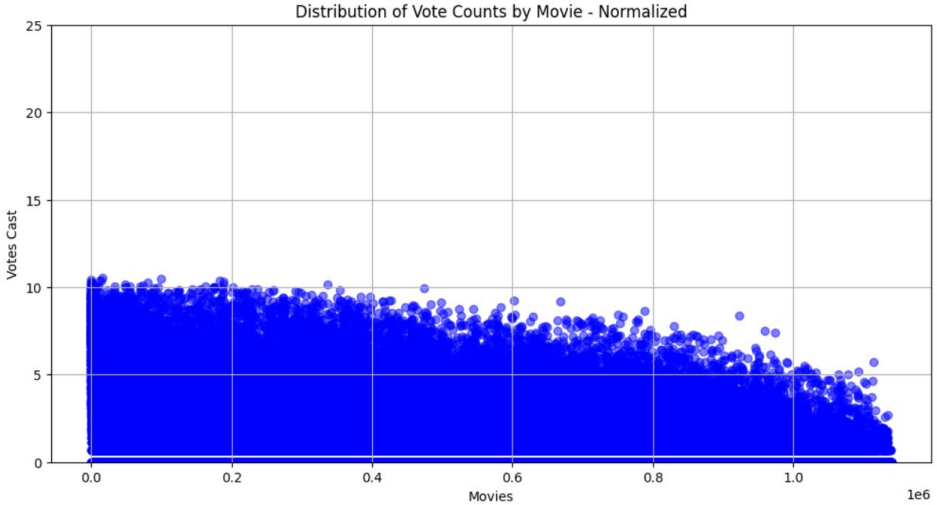




SQL Classification

- Average of all votes ~21.6
- Wide gap in vote count distribution
- Normalize the data
- Decide what will be the floor of a High, Medium, and Low vote count



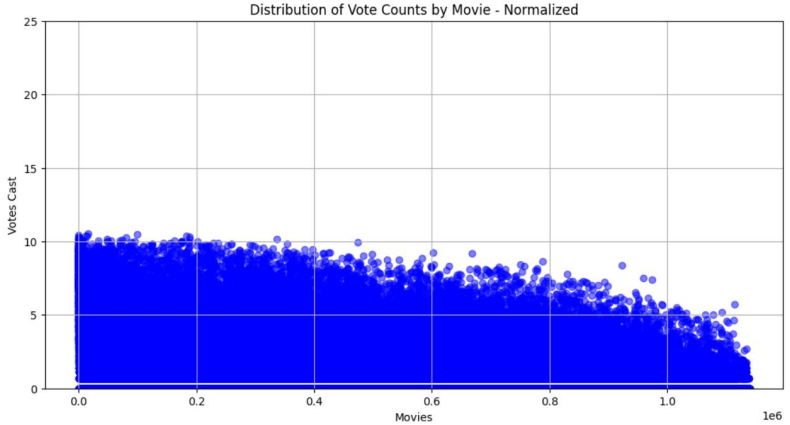


SQL Classification

- Find mean average of all vote counts
- Take the log of the mean avg. multiplied by 10 the get the floor of the “High” classification
- Take the log of the mean avg. multiplied by 2 to get the floor of the “Medium” classification

Let vote_count = x Let avg_count = y

$$\sum_{i=1}^{n} \begin{cases} \text{'High'} & \text{if } \ln(\text{NULLIF}(x_i, 0)) \geq \ln(y \times 10) \\ \text{'Medium'} & \text{if } \ln(\text{NULLIF}(x_i, 0)) \geq \ln(y \times 2) \\ \text{'Low'} & \text{else} \end{cases}$$



Avg. Vote Counts =

```
print(df['vote_count'].mean())
```

```
21.576217783652183
```

Conditionals (if)

High ≥

```
print(np.log(mean_vc * 10))
```

```
5.374176772572347
```

Medium <

Medium ≥

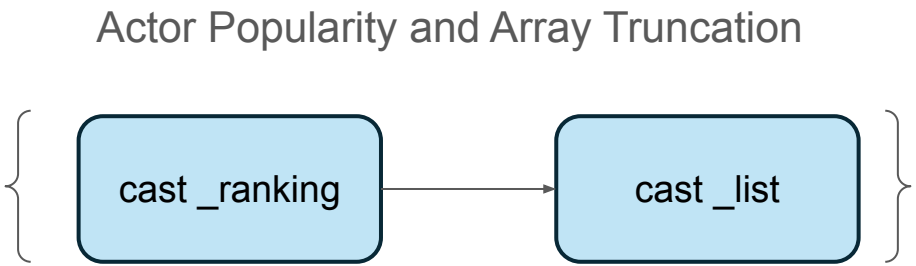
```
print(np.log(mean_vc * 2))
```

```
3.7647388601382463
```

Low <

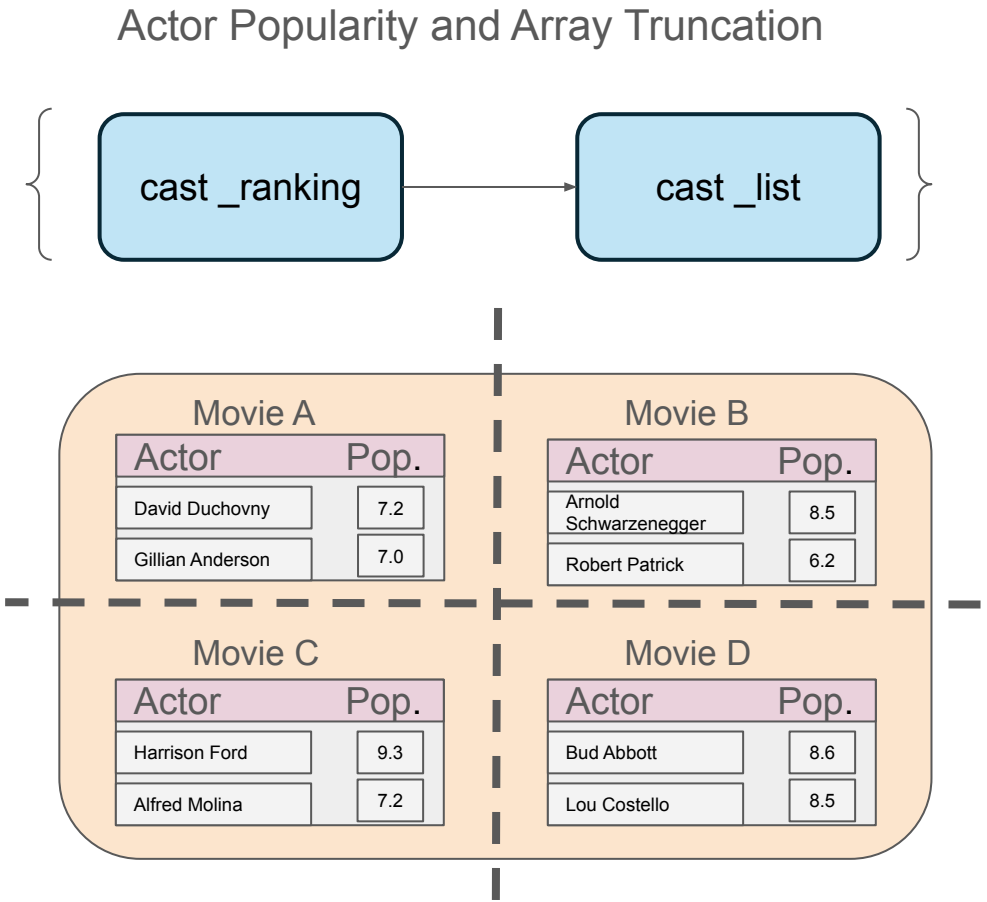
SQL Ranking

- Cast Ranking will rank the cast members within each movie
- Cast List will create an array of the ranked cast members
- Filter the output to only include the top 5 most popular cast members



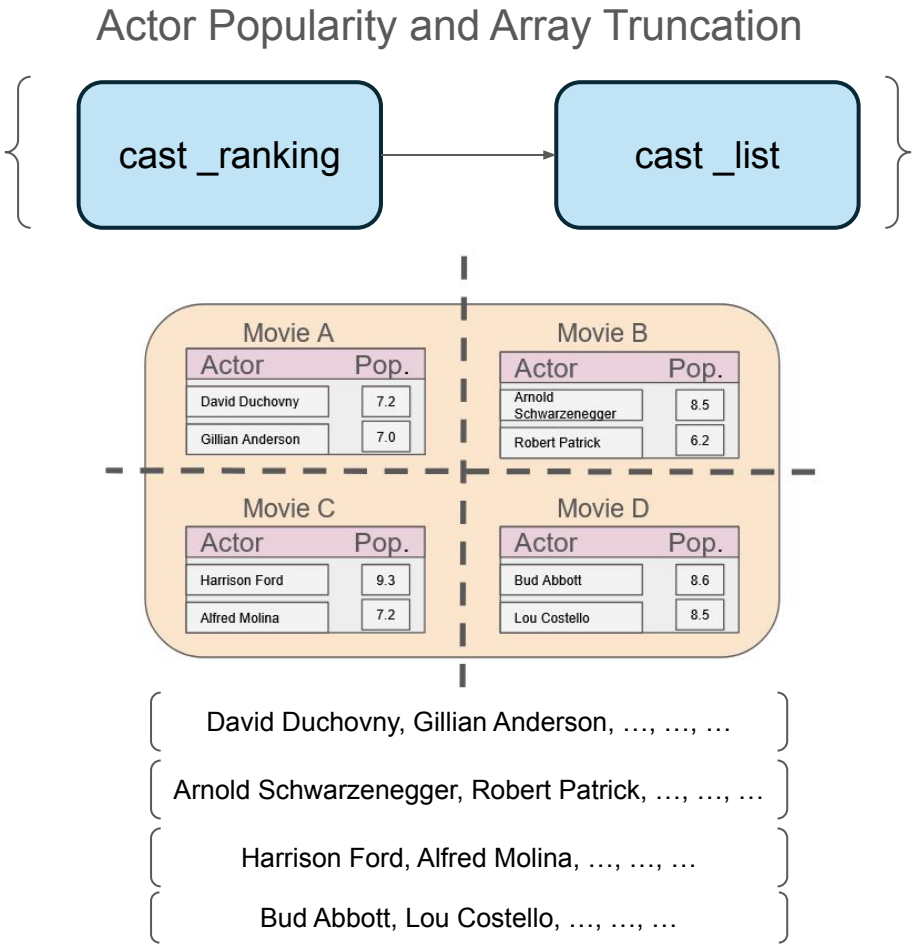
SQL Ranking

- Partition the actors by each movie
- Within each movie order cast members by popularity



SQL Ranking

- Partition the actors by each movie
- Within each movie order cast members by popularity
- Filter for the top 5 ranked actors for each movie
- Output an Array



SQL Query Output

- Select title, classified viewer score, vote average, vote classification, normalized vote count, top 5 actors, and movie overview
- Subquery to return specified Genre and language
- Order movies by classification score and vote average.

Movie	Viewer score classification	Vote Average	Vote classification	Vote_count	Movie_cast	Overview
The Dark Knight	High	8.5	High	10.41711923563258	['Gary Oldman', '陳冠希', 'Cillian Murphy', 'Morgan Freeman', 'Michael Caine']	Batman raises the stakes…
The Lord of the Rings: The Return of the King	High	8.5	High	10.114720452824503	['Cate Blanchett', 'Viggo Mortensen', 'Elijah Wood', 'Sean Bean', 'Ian McKellen']	As armies mass for a final battle…
Inception	High	8.4	High	10.52023953184236	['Leonardo DiCaprio', 'Tom Hardy', 'Cillian Murphy', 'Tom Berenger', 'Joseph Gordon-Levitt']	Cobb, a skilled thief who commits corporate espionage…
The Lord of the Rings: The Fellowship of the Ring	High	8.4	High	10.15085531407452	['Cate Blanchett', 'Viggo Mortensen', 'Elijah Wood', 'Sean Bean', 'Ian McKellen']	Young hobbit Frodo Baggins, after inheriting a mysterious ring…
The Lord of the Rings: The Two Towers	High	8.4	High	10.00946787607336	['Cate Blanchett', 'Viggo Mortensen', 'Elijah Wood', 'Ian McKellen', 'Liv Tyler']	Frodo Baggins and the other members of the Fellowship…

SQL Query Output

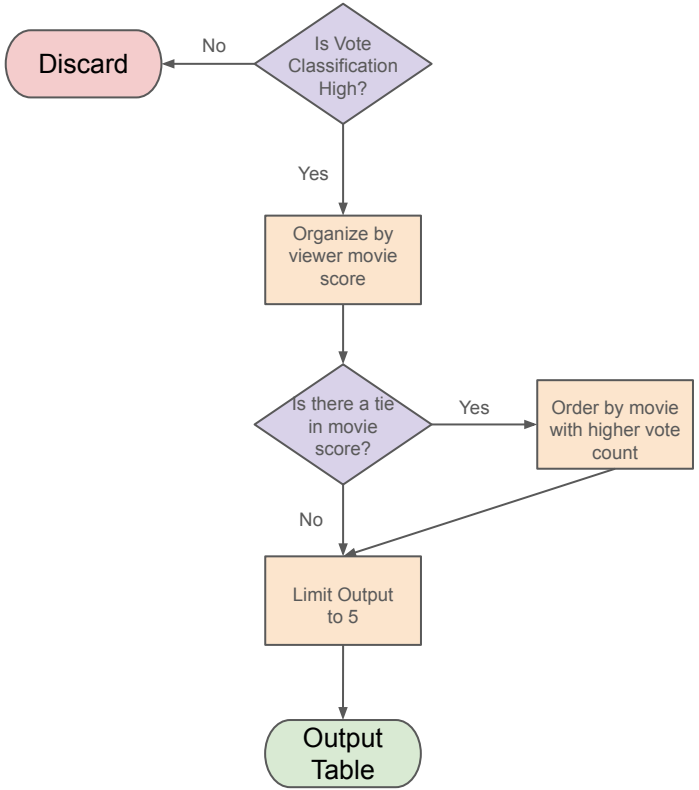
- Select title, classified viewer score, vote average, vote classification, normalized vote count, top 5 actors, and movie overview
- Subquery to return specified Genre and language
- Order movies by classification score and vote average.

Movie	Viewer score classification	Vote Average	Vote classification	Vote_count	Movie_cast	Overview
The Dark Knight	High	8.5	High	10.41711923563258	['Gary Oldman', '[illegible]', 'Cillian Murphy', 'Morgan Freeman', 'Michael Caine']	Batman raises the stakes...
The Lord of the Rings: The Return of the King	High	8.5	High	10.1147204452824503	['Cate Blanchett', 'Viggo Mortensen', 'Elijah Wood', 'Sean Bean', 'Ian McKellen']	As armies mass for a final battle...
Inception	High	8.4	High	10.52023953184236	['Leonardo DiCaprio', 'Tom Hardy', 'Cillian Murphy', 'Tom Berenger', 'Joseph Gordon-Levitt']	Cobb, a skilled thief who commits corporate espionage...
The Lord of the Rings: The Fellowship of the Ring	High	8.4	High	10.15085531407452	['Cate Blanchett', 'Viggo Mortensen', 'Elijah Wood', 'Sean Bean', 'Ian McKellen']	Young hobbit Frodo Baggins, after inheriting a mysterious ring...
The Lord of the Rings: The Two Towers	High	8.4	High	10.00946787607336	['Cate Blanchett', 'Viggo Mortensen', 'Elijah Wood', 'Ian McKellen', 'Liv Tyler']	Frodo Baggins and the other members of the Fellowship...

```
75      WHERE EXISTS (
76          SELECT 1
77          FROM JSONB_ARRAY_ELEMENTS(m.genres) AS genre
78          WHERE (genre ->> 'id')::INTEGER = {GENRE_ID} -- placeholder for genre id
79      )
80      AND m.original_language = '{LANGUAGE_PARAM}'
```

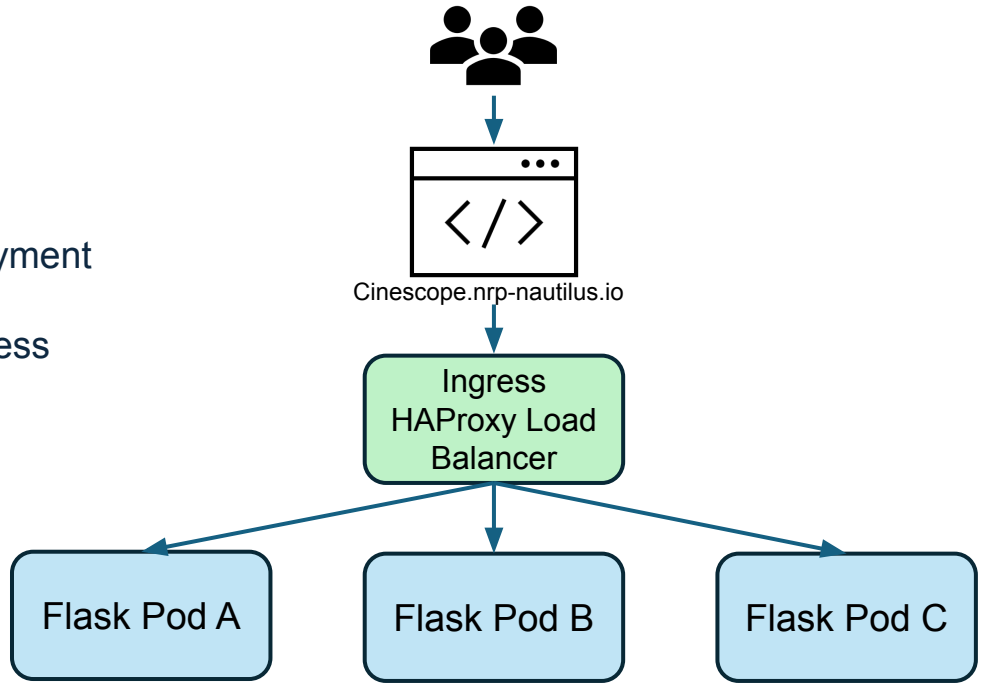
SQL Query Output

- Ordering The Output
  - Check that the movie has a “High” vote count classification.
  - Organize “High” movies by their average score
  - In cases where there is a tie, list the movie with the higher vote count in descending order.
  - Output the table.

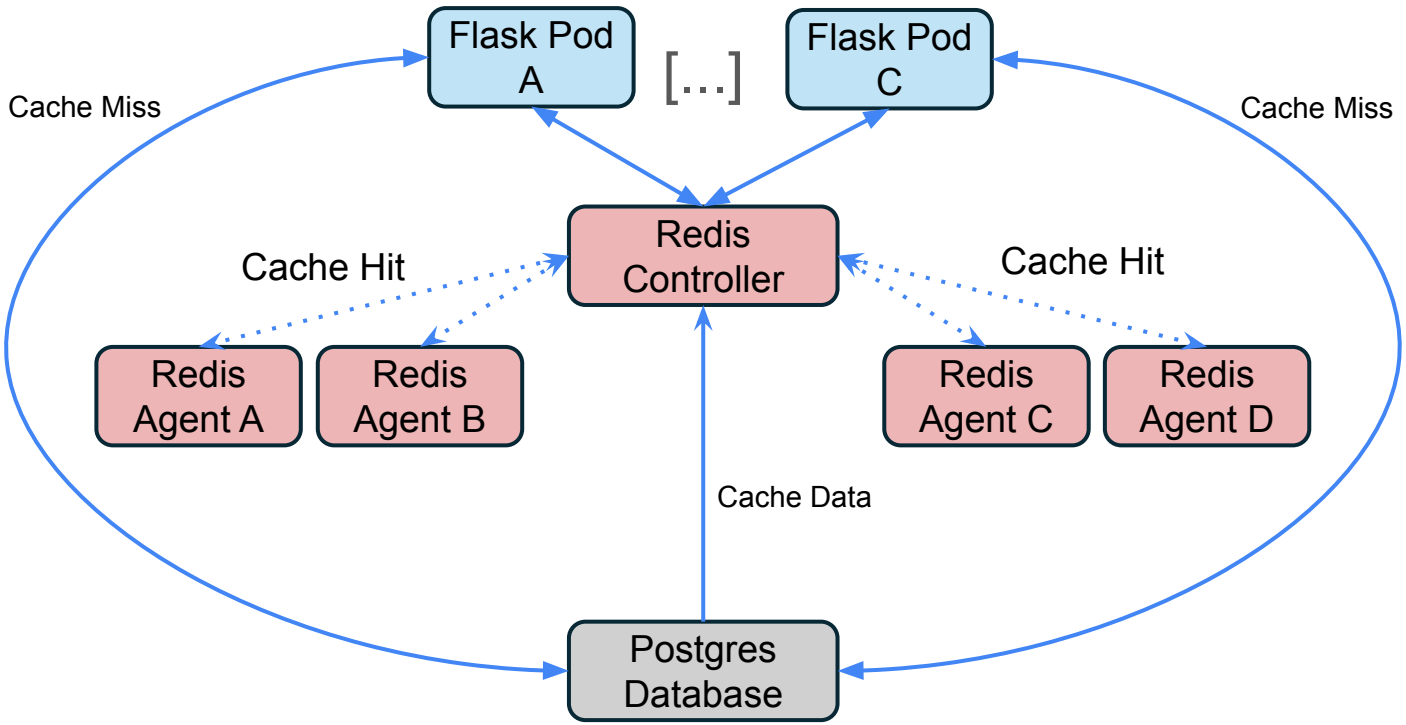


Horizontal Scaling

- Flask Pods have been created as a deployment with a 3 x replication
- As a End-user visits the webpage the ingress controller directs traffic

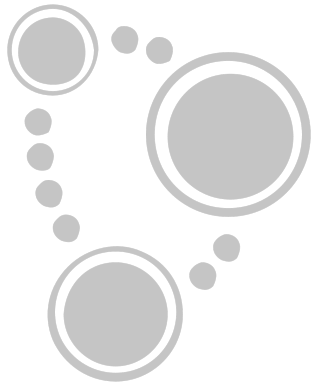


Horizontal Scaling



Neo4j in Cinescope:
Enabling Graph-Based Analysis

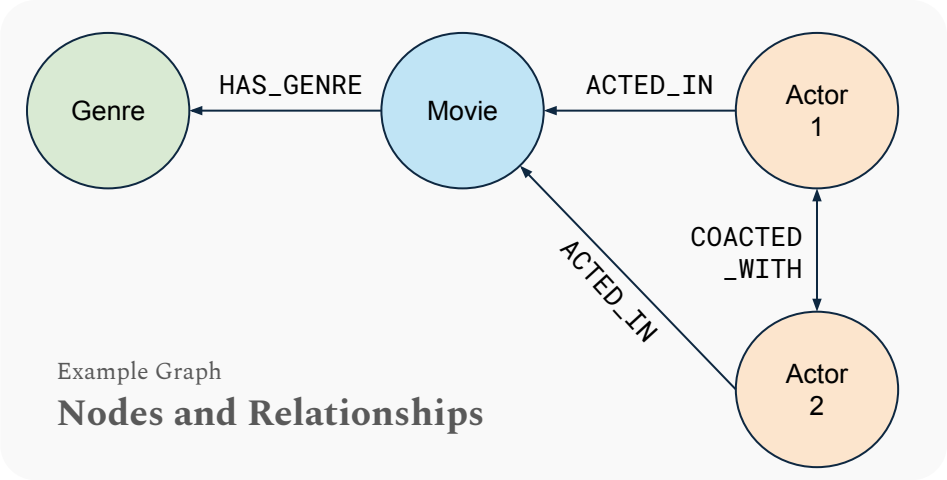
- Leverages graph database for **relationship exploration**
- **Complements PostgreSQL** for efficient data analysis
- **Supports** fast traversal of movies, genres, and actors
- **Integrated** into a microservices architecture via **Flask**



Data Modeling & System Integration

- **Nodes:**

  Genre, Movie, Actor
- **Relationships:**
  - ACTED_IN (Actor → Movie)
  - HAS_GENRE (Movie → Genre)
  - COACTED_WITH (Actor ↔ Actor)
- **PostgreSQL** supplies top movies based on SQL queries
- **Flask** routes coordinate data retrieval and graph construction



Implementation & Query Techniques

- Batch data import to CSV conversion
- Use of MERGE to prevent duplicate nodes
- Efficient Cypher queries:
  - Retrieve top movies per genre
  - Build actor co-acting networks
- Filtering on popularity to manage large datasets

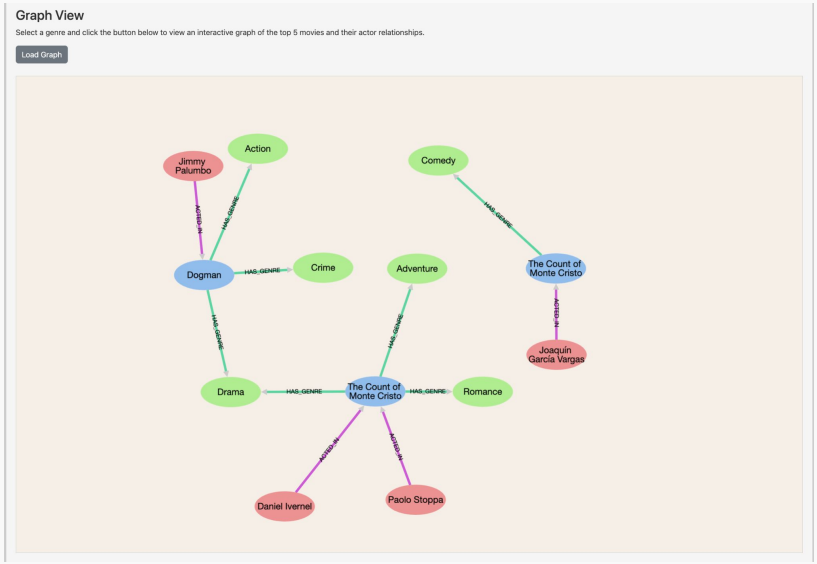
Example Cypher Query

Finds pairs of actors who have co-starred in more than one movie within the specified genre

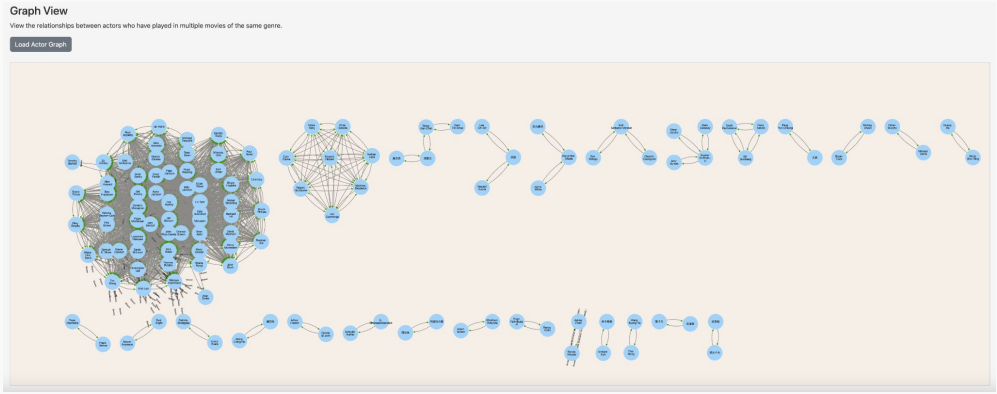
```
MATCH (a1:Person)-[:ACTED_IN]->(m:Movie)<-[:ACTED_IN]-(a2:Person),
      (m)-[:HAS_GENRE]->(g:Genre {name: $genre})
WHERE a1 <> a2
WITH a1, a2, COUNT(m) AS shared_movies
WHERE shared_movies > 1
RETURN a1.name AS Actor1, a2.name AS Actor2, shared_movies;
```

Interactive Graph Visualization

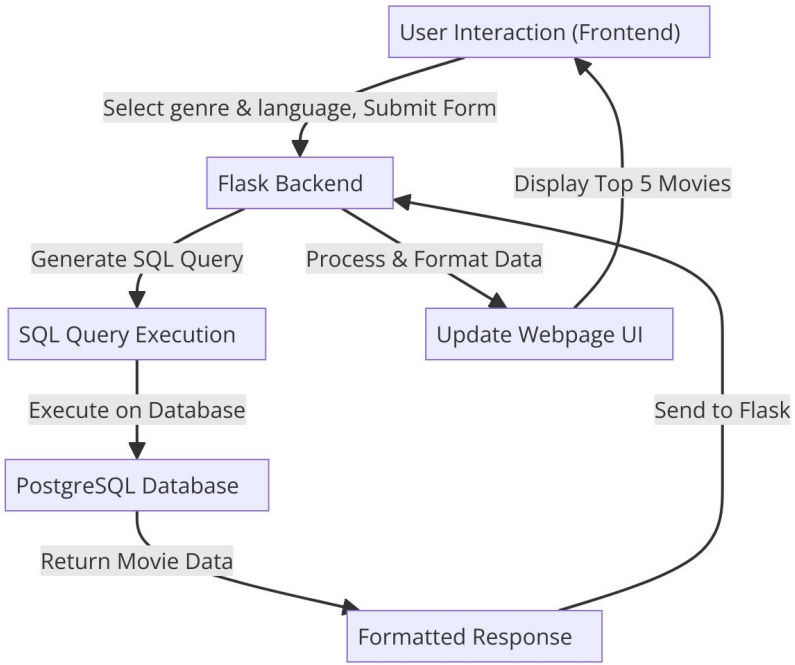
- Two **graph views**
- Rendered using Cytoscape.js
- Enhances data exploration and user engagement
- Opportunities for future interactive features



← **Graph 1:** Top Movies & Their Actors
↓ **Graph 2:** Actor Co-acting Network



Front End development & PostgreSQL Integration



The **Front-End** captures user input and sends requests to the backend.

Flask (Back-End) processes the request, generates SQL queries, and interacts with the database.

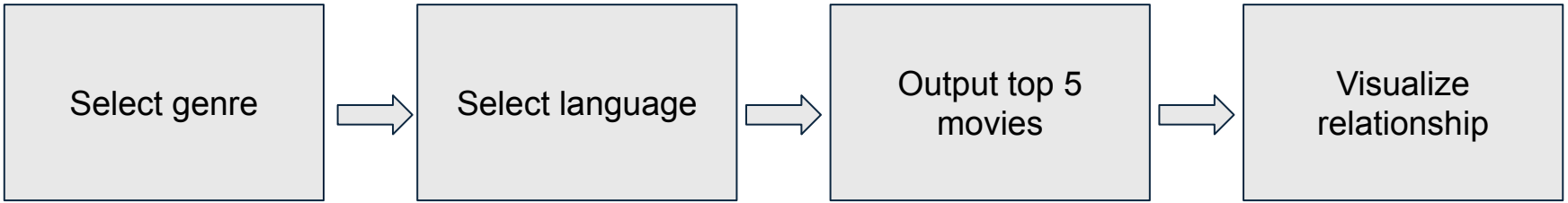
PostgreSQL executes queries and returns relevant data.

Flask formats the response and updates the **Front-End** dynamically.

The result is a **seamless user experience**, displaying movie recommendations in real time.

Front End development & PostgreSQL Integration

- 1 User selects genre & language (Dropdowns & UI Forms)
- 2 Flask retrieves corresponding genre ID from PostgreSQL
- 3 SQL Query fetches **top 5** most popular movies per selection
- 4 Results dynamically rendered in a structured HTML table



Front End development & PostgreSQL Integration

- Flask is used to tie together the back-end and front-end.
- Handles GET requests to display genres and languages.
- Processes POST requests to filter movies based on user selections.
- Jinja2 templating integrates dynamic content into HTML pages.

•• Flask acts as a bridge between user interactions & database queries.

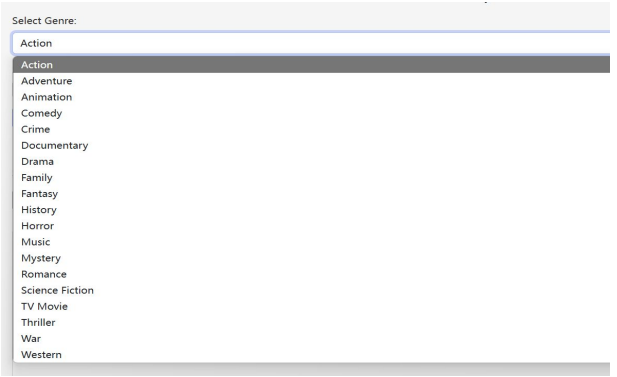
•• **Psycopg2** is used for secure SQL query execution.

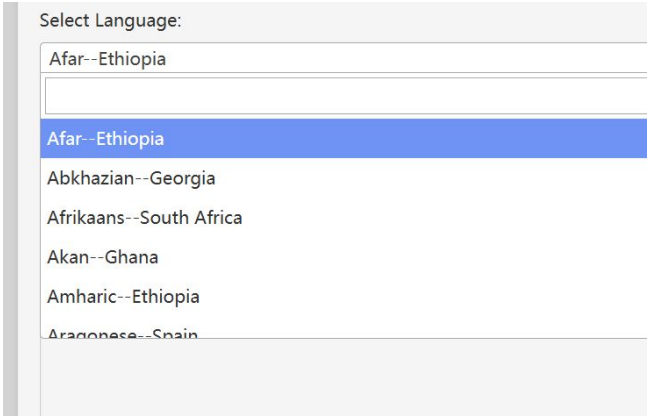
•• Queries dynamically filter **movies by genre & language**.

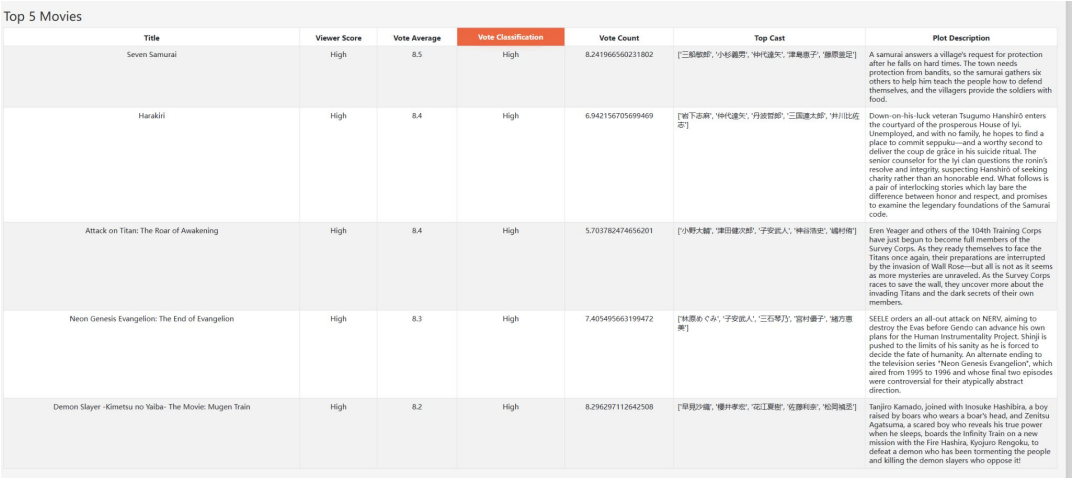
•• Results are sent back to Flask, formatted with Jinja2 templates.

Front End development & PostgreSQL Integration

- ✔ Flask (Backend Web Framework)
- ✔ HTML, CSS, Bootstrap (UI/UX Design)
- ✔ Jinja2 (Dynamic Content Rendering)
- ✔ Select2 & JavaScript (Enhanced User Interactions)
- ✔ PostgreSQL (Data Storage & Querying)







Thanks for your Listening!

DSC 202 Final Project
Cinescope: Scalable Movie Genre Search and Analysis

Group #18 Joel Polizzi, Dongting Cai, Xuanwen Hua
March 18, 2025