**IMDB Architecture Document**

Input REST api’s from outside world hits Katran first

DB

**Note**: Katran and Nginx are not implemented / used in this project but its part of this architecture document keeping network scalability in mind. Rest all is implemented.  
  
**L4 Forwarding plane using Katran:**  
-----------------------------------------------  
Why an L4 Load Balancer is required ?   
In all typical CRUD operations in real world (99% of web sites), the size of output data is magnitude of times larger than the size of input data. In other words egress is much larger than ingress for most of the web sites. If all egress (out bound traffic) passes through load balancer then the load balancer itself becomes a bottleneck. So, the best approach is to keep the outbound traffic (egress) path not same as ingress path. This can be achieved by having a separate Forward plane and implementing DSR (Direct Server Return). Katran is an L4 Forwarding plane open sourced by Facebook.   
  
1. Katran should be installed along with each apigateway micro service.  
2. Katran reads the tcp packet and performs consistent hashing using (sourceIP, source port, destinationIP, destination port, protocol) and selects the backend (apigateway) which in turn does DSR (direct server return) and directly responds to client. This evenly distributes our outbound egress network traffic on each front end servers.  
  
  
More details on Katran can be found using below links:  
<https://github.com/facebookincubator/katran>  
<https://engineering.fb.com/open-source/open-sourcing-katran-a-scalable-network-load-balancer/>  
  
**Api Gateway Micro Service:**  
--------------------------------------  
1. The client’s tcp connection ends here on apigateway micro service.  
2. Apigateway is the entry point to our application and here it does following:  
 a. Authenticate for login functionality, this is required only for adding “comments”, “rating” movie and   
 admin authentication is required for adding “movie”.  
 b. This also acts as a “Aggregator“ micro service, collecting search results and recommendation from   
 “search micro service” and “recommendation engine micro service” and then pulling respective   
 movie comments from “useraction micro service”.  
3. Once apigateway starts up it reads all usernames from its database and stores into scalable bloom   
 filter, so that during login in if the scalable bloom filter says username is not present then we avoid   
 hitting the database and return Unauthorized user immediately. This scalable bloom filter can help us   
 when under attack and offload the database.  
4. Since there are multiple apigateway micro services, bloom filters in each micro services should have   
 the recently added username. We, use Apache Kafka as message broker to send the just created new   
 username across all api gateways micro services to update their bloom filter. There is no api to create   
 a new User but the message broker part is implemented.  
5. You can scale this micro service horizontally, since these do not contain any state information they   
 can be added or removed anytime after notifying the Katran L4 forwarding plane.  
6. All apigateway micro services should have same virtual IP to implement DSR.  
  
  
**Search micro service:**  
------------------------------  
1. Why “Search” is a separate micro service?  
 a. Search requires complex data structure and algorithms and thus requires a dedicated team to   
 improve the search results over time. So, we create a lose coupling of Search with rest of   
 application logic, hence Search should be a separate micro service.  
 b. Search algorithm right now only depends on Movie Name but in future we may want to take   
 other Movie parameters like “Rating”, keywords in “Description” etc to improve the result. Hence,   
 a separate copy of database for search micro service is required.  
2. Prefix Search can be done directly using database using LIKE keyword.  
 eg: SELECT MovieId, Name FROM Movie WHERE Name LIKE ‘Twil%’;  
 If the user does not type the prefix in correct sequence then “Twilight” will not be returned. So the   
 “levenshtein edit distance” is implemented instead.  
3. The output of Search is arranged by increasing order of edit distance.  
4. The output of Search micro service is responded back to apigateway which then pulls the respective   
 movie comments from “user action micro service”.  
5. This micro service also polls on “TOPIC-Movie” topic on Kafka to consume and store in its DB the   
 newly added movie by admin which is sent by “useraction” micro service to kafka.  
  
  
**Recommendation Engine micro service:**  
------------------------------------------------------  
1. Why “Recommendation” Engine is a separate micro service?  
 a. Recommendation requires complex data structure and algorithms and thus requires a dedicated   
 team to improve the results over time. So, we create a lose coupling with rest of   
 application logic, hence it should be a separate micro service.  
 b. The landing page of imdb is the output of Recommendation Engine and this output may also take   
 the gps (latitude, longitude) into consideration when displaying Movies on Landing Page. Over a   
 period of time, this is bound to get complicated, so having a loose coupling with rest of business   
 logic is good decision.  
2. Right now the “/home” page (landing page) is the output of recommendation engine.  
3. Right now we just display the output in decreasing order of Movie Rating.  
4. The output of Recommendation engine is returned back to apigateway which intern gets all relevant   
 comments from “useraction” micro service.  
5. This micro service also polls on “TOPIC-Movie” topic on Kafka to consume and store in its DB the   
 newly added movie by admin which is sent by “useraction” micro service to kafka.  
  
**User Action micro Service**:  
------------------------------------  
1. Why “User Action” is a separate micro service?  
 a. This micro service will always have just plain CRUD operation, the size of database (store all user   
 actions rating, comments) can be magnitude of time larger than the actual Movie data. Hence, this   
 size of data scales differently.  
2. When admin user creates a new movie, this micro service publishes the data to Kafka which intern is   
 read by Search micro service and Recommendation micro service in order to keep the Movie data in   
 sync with each other.  
3. This micro service stores all the rating and comments added by individual user.  
4. This micro service also returns all the actions (Rating, Comments) on movies performed by single user.  
5. Since the average movie data is pre populated we don’t have rating of all past users who voted, so if you rate a movie, your individual rating is saved in useraction but that will not be reflected in average movie rating since its not possible to calculate average movie rating in this case.

Movie created by admin has to be copied to all databases

Katran  
XDP eBPF

Katran  
XDP eBPF

Katran selects backend (apigateway) which does DSR

DB

DB

User Action micro Service

Recommendation engine micro service

Search micro service

Nginx

Nginx

Nginx

Apache Kafka

Api Gateway  
micro service

Api Gateway micro service

Future improvements to Consider:  
------------------------------------------------  
1. For communication of apigateway to outside world we should always use REST api’s, but communication of apigateway to internal micro services we should consider using Apache   
Thrift instead of internal REST api’s, hence have to replace Nginx with HaProxy. This is in order to improve performance (response time) and resource usage.