Community Application Stream Processing Analytics

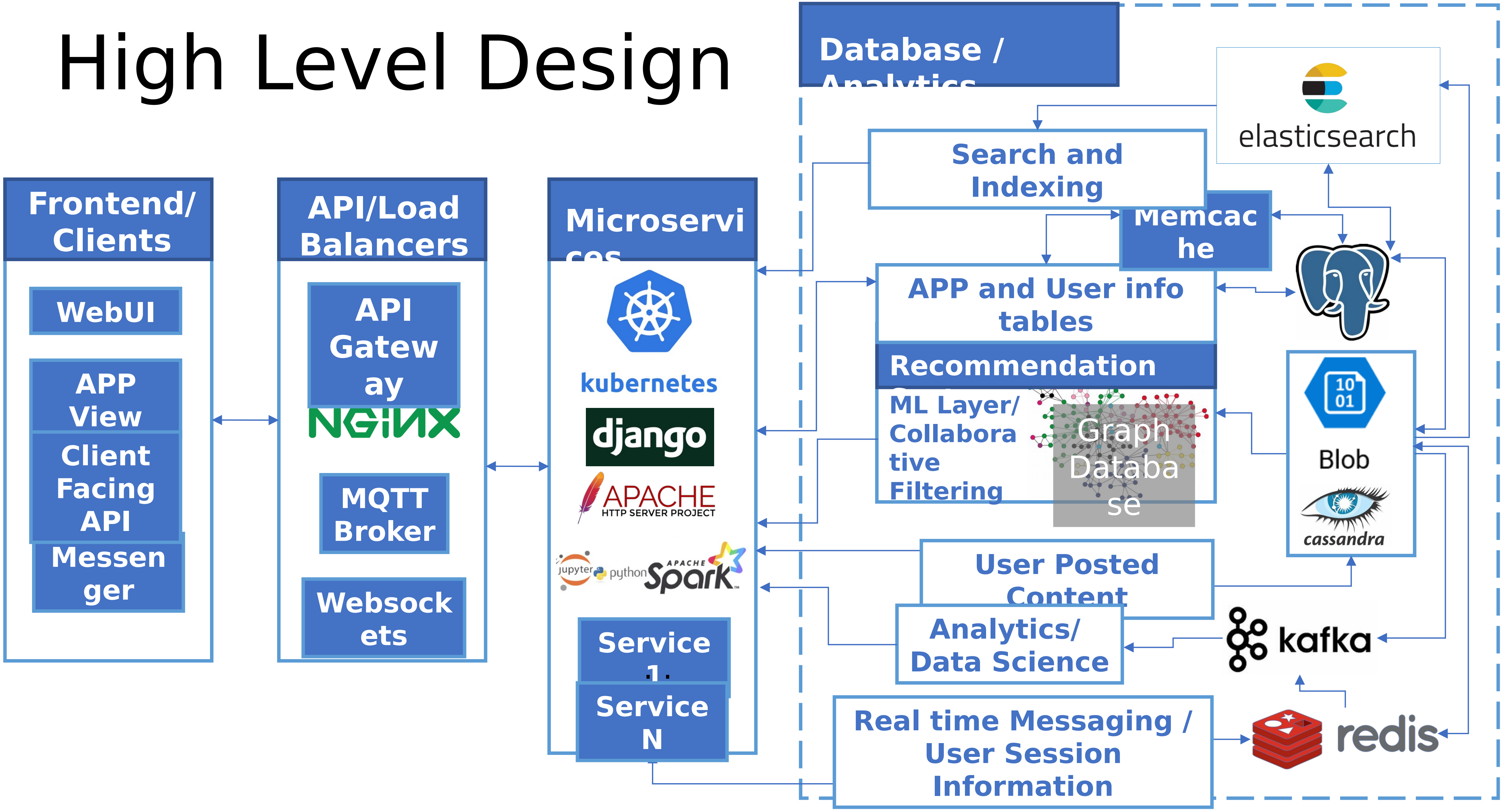
Raza Abbas (2019ad04095)

Sandeep Kumar (2019ad04106)

Shanur Rahman (2019ad04065)

Explanation Video URL
Part 1: https://youtu.be/PosptjOpyr8
Part 2: https://youtu.be/RLEfJ-aNLas

High Level Design



Frontend/ Clients

WebUI

APP View

Client Facing API

Messenger

API/Load Balancers

API Gateway

MQTT Broker

Websockets

Microservices

Service 1

Service N

Database / Analytics

Search and Indexing

Memcache

APP and User info tables

Recommendation

ML Layer/ Collaborative Filtering

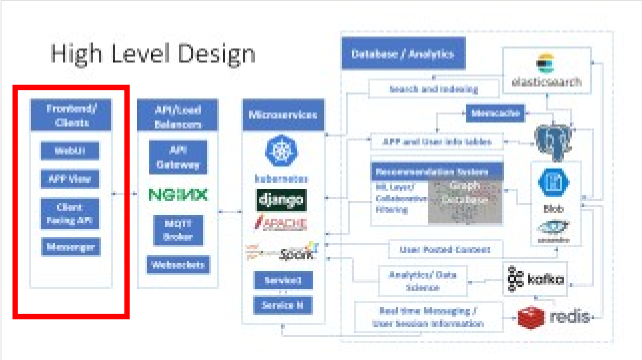
Graph Database

User Posted Content

Analytics/ Data Science

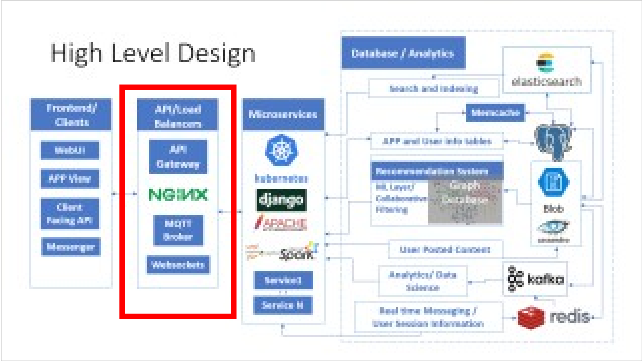
Real time Messaging / User Session Information

Frontend / Client Tier



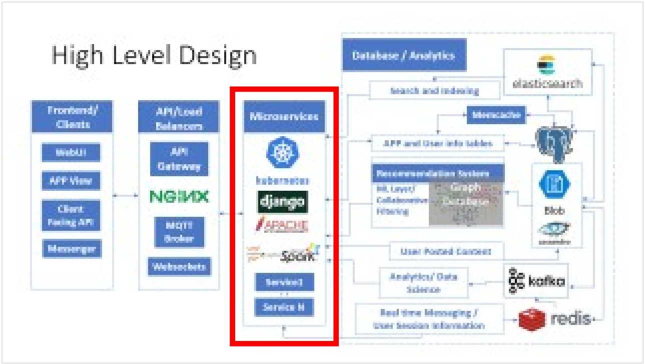
- These are the only supported access points for users w.r.t application.
- Includes:
  - WebUI
  - Application View
  - Messenger
  - Client Facing API

API / Load Balancer(s) Tier



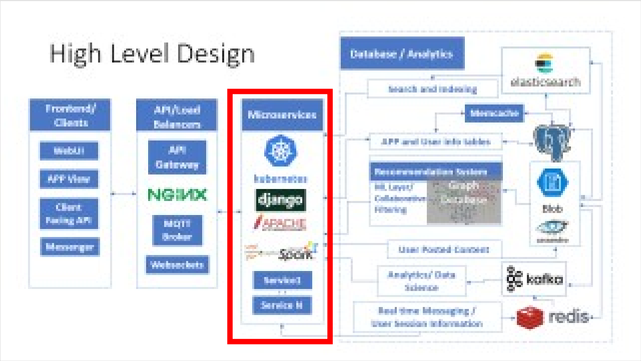
- API Gateway tier aggregates various microservices based on their protocol and function, to multiple APIs.
- Type of API gateways:
  - Rest API based on HTTP(s) with supported operations such as get, post, put.
  - Websockets to support real time chat applications.
  - MQTT to support routing and logging of high speed incoming messages to database and other applications. User chats, logs and maintenance.
- API gateway also acts as a firewall, delegating authentication to an Internal microservice or API.

Micro Services Tier



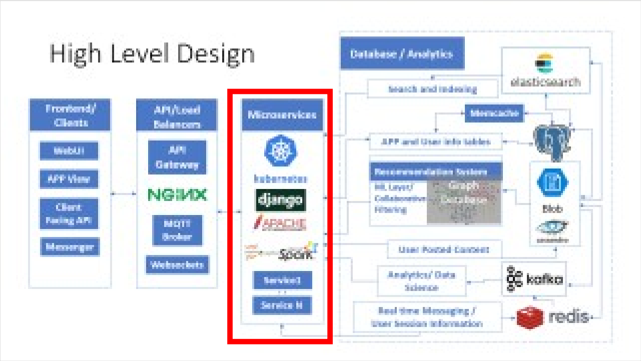
- Forms the basic application business logic.
- Every interaction to application goes through microservices.
- These services will write or read content from databases directly or through data models. Such as Django data models.
- For frontend UI Django is an ideal choice. This will contain app routing and user authentication logic.
- For Backend services Django Rest is an ideal choice. This will contain the queries and routines to be executed in analytics tier.
- For serving static resources apache server will be used.

Micro Services Tier



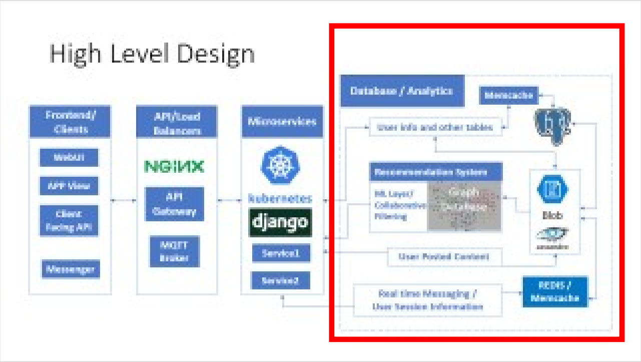
- Based on Kubernetes cluster(s).
- Each service group contains multiple separate services.
- Will have both:
  - Sync microservices: Data Access, media, routing, authentication, etc.
  - Async microservices: Background and batch processing.
- Authentication within microservices and external requests is based on User session information stored in a REDIS cache database. It will be exposed as an API(Oauth type).
- Each microservice has separate compute / network resources, the stack can be scaled by adding more instances (Horizontally scalable).
- A service bus will be deployed for inter services communication.

Micro Services Tier



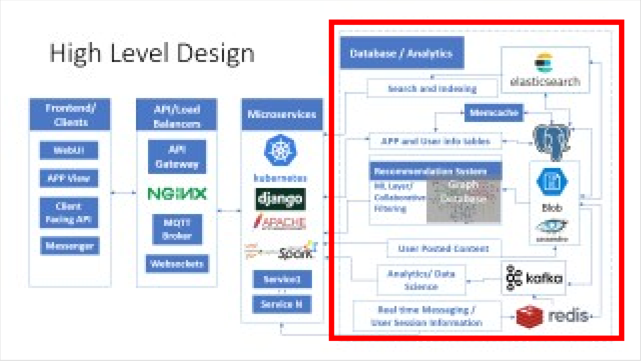
- Microservice will also maintain and manage application definition involving:
  - Application scaling
  - Application security
  - Application management
  - Anomaly detection
- One service group will be dedicated to above said functions.

Database Tier



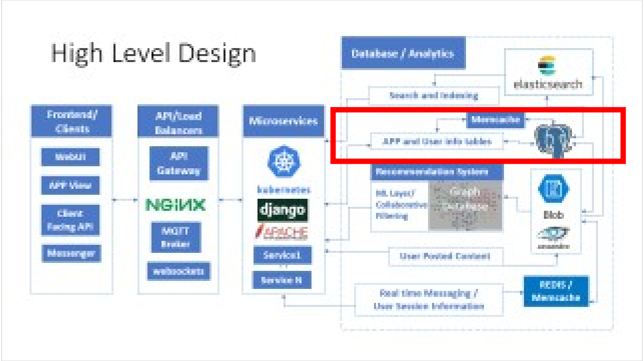
High Level Design

Types of data streams



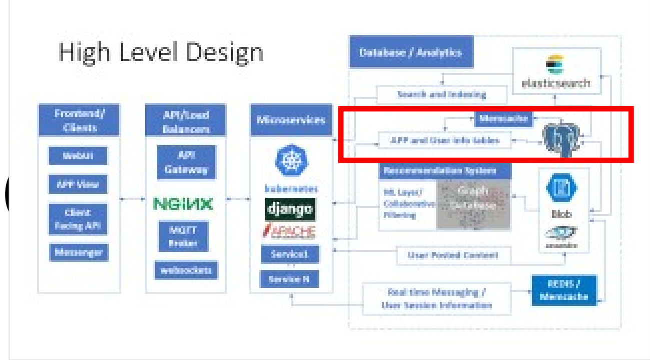
- APP and User info tables
  - Structured data
- User posted content
  - Unstructured/structured data
- Real time Messaging / User Session Information
  - Unstructured/structured data

APP and User info tables



- Contains user definition information, app definition information, user authentication information and regulatory data. These are stored in form of multiple simple tables.
- Some of the tables would be generated using Django models.
- As the new users sign up a moderately sized database will build up.
- Data is accessed and updated infrequently but high consistency is a requirement. Postgresql is an ideal choice based on the requirements.

APP and User info tables sch

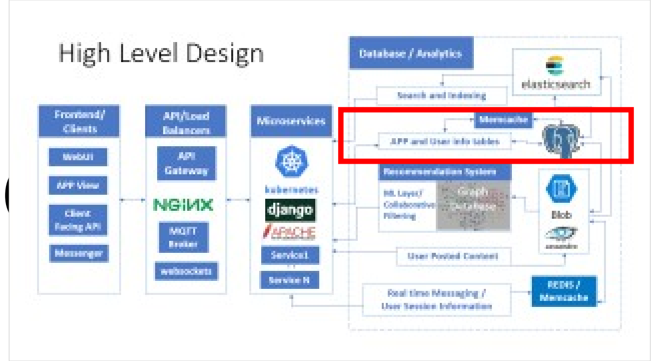


Format

- *table_name*
  - *column_name:datatype:indextype*

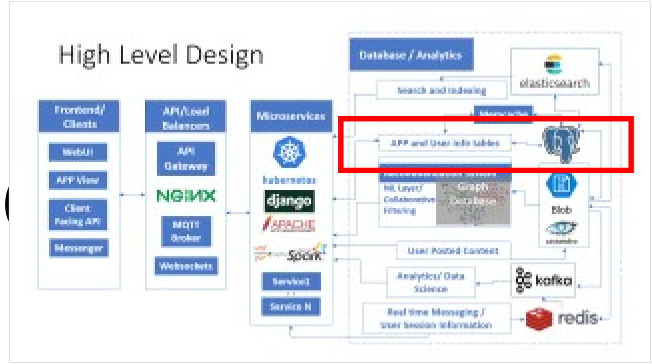
- user_master
  - user_uuid:string:btree
  - user_email:string:btree
  - user_name:string:btree
  - user_phone:string
  - user_signup_date:datetime
  - user_last_logged_in:datetime
  - user_password_hashed:string

APP and User info tables sch



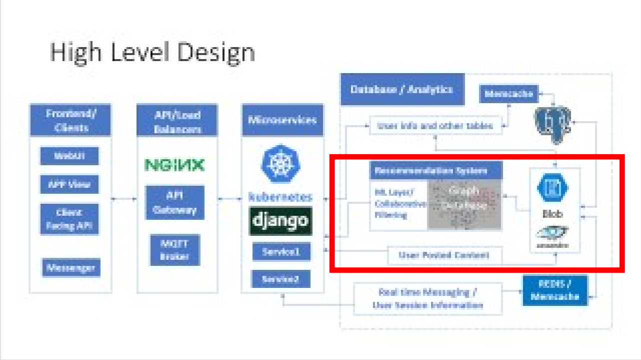
- user_group_master
  - group_uuid:string
  - group_name:string
  - group_user_count:int
  - group_is_active:Boolean
  - group_created_on:datetime
  - group_deleted_on:datetime
- user_group_user_list
  - group_user_index:string
  - user_uuid:string
  - user_name:string
  - is_active:boolean:btree
  - is_admin:boolean
  - added_on:datetime
  - removed_on:datetime

APP and User info tables sch



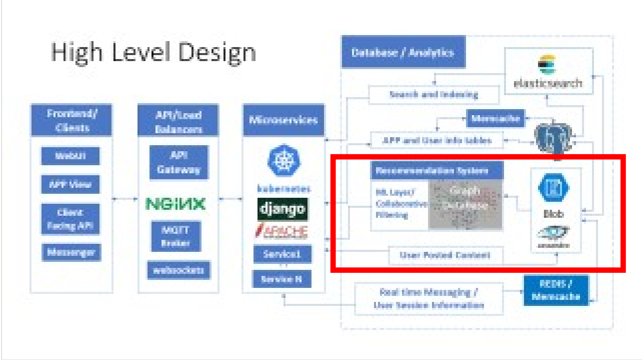
- application_master
  - app_pods_count:int
- app_service_groups_master
  - service_uuid:string
  - service_pods_count:int
  - service_image_id:string

User posted content



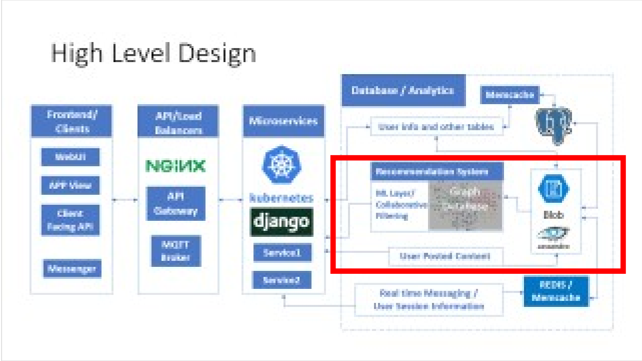
- User activity such as posts, comments, likes, shares, messages archival, etc.
- This has all three essence of big data:
  - High Velocity
  - High Volume
  - High Variety
- Here data is:
  - Accessed frequently
  - Inserted frequently
  - Updated infrequently
- Highly reliable and Highly available system is required, immediate consistency is not required.
- Cassandra is an ideal choice for storing user activity in form of text, small blobs and pointers to big blobs.
- Blob Storage for storing big blobs, with a pointer stored in Cassandra to maintain metadata and other activity on the item.

User posted content
Why Cassandra ?



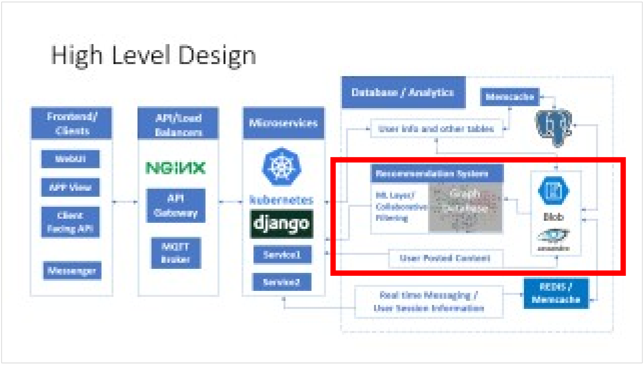
- Cassandra is an ideal choice for storing user activity in form of text, small blobs and pointers to big blobs.
- NoSql format provides flexibility to store complicated user activity data easily.
- The datastore is highly available and highly reliable by replicating across various nodes.
- The datastore will be not be immediately consistent but eventually consistent. That means updates are slow and it takes time to propagate data updates across nodes.
- Casandra will also support ongoing analytics effort for our use case.

User posted content nosql schema



- post_master
  - post_uuid:string
  - post_name:string
  - post_author_uuid:string
  - post_text:string
  - post_pointers:json
  - post_url_slug
  - post_scope
  - post_share_count
  - post_comment_count
  - post_reaction_count
  - post_comments
    - comment_timestamp
    - comment_user
    - comment_text
  - post_reactions
    - reaction_timestamp
    - reaction_user
    - reaction_code

User posted content nosql schema



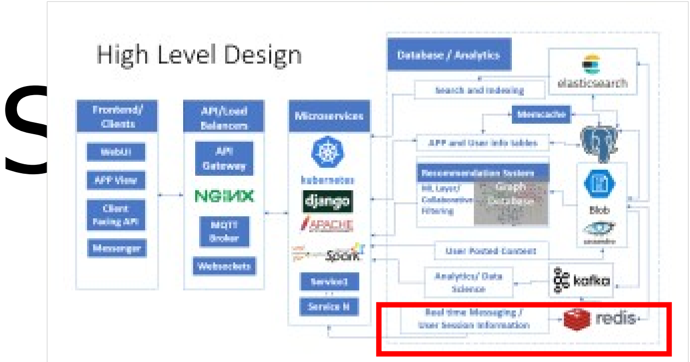
- share_master
  - post_id
  - author_user_id
  - share_user_id
  - share_scope
  - is_forwarded
  - Is_reshared
  - share_comment_count
  - share_reaction_count
  - share_comments
    - comment_timestamp
    - comment_user
    - comment_text
  - share_reactions
    - reaction_timestamp
    - reaction_user
    - reaction_code

User posted content nosql schema

High Level Design
Frontend/ Clients
WebUI
APP View
Client Facing API
Messenger
API/Load Balancers
NGINX
API Gateway
MQTT Broker
Microservices
kubernetes
django
Service1
Service2
Database / Analytics
Memcache
User info and other tables
Recommendation System
ML Layer/ Collaborative Filtering
Graph Database
Blob
cassandra
User Posted Content
Real time Messaging / User Session Information
REDIS / Memcache

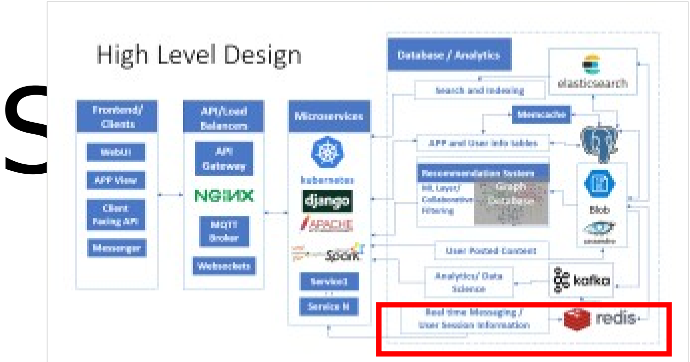
- messages_archive
  - message_uuid:string
  - user_uuid:string
  - message_timestamp:datetime
  - destination_group_uuid:string
  - destination_user_uuid:string
  - contains_media:Boolean
  - Media_pointer:string
  - reply_list
    - reply_message_uuid

Real time Messaging / User S Information



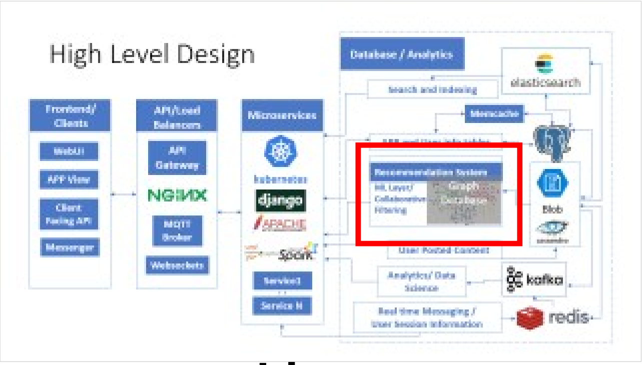
- Incoming user messages and user session information is temporarily stored in a highly available REDIS cache. Eventually the data is made persistent in Cassandra for archival and analytics.
- These messages are transmitted back to subscribers in form of web socket push. Author and subscribers can be:
  - One to one (Direct messages)
  - One to many (Group messages)
- Blobs shared in form of media are treated the same way as user posted content. Actual data is delegated to that microservice, but a pointer is stored in chat to be consumed by UI and an integrated experience is served to users.

Real time Messaging / User S Information



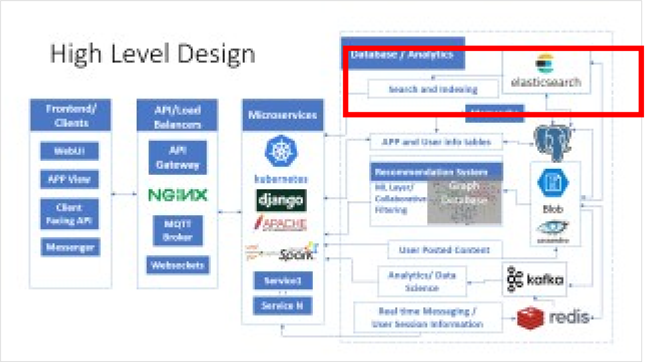
- message_master
  - Message_uuid:string
  - user_uuid:string
  - message_timestamp:datetime
  - destination_group_uuid:string
  - destination_user_uuid:string
  - contains_media:Boolean
  - Media_pointer:string
  - reply_list
    - reply_message_uuid

Recommendation System



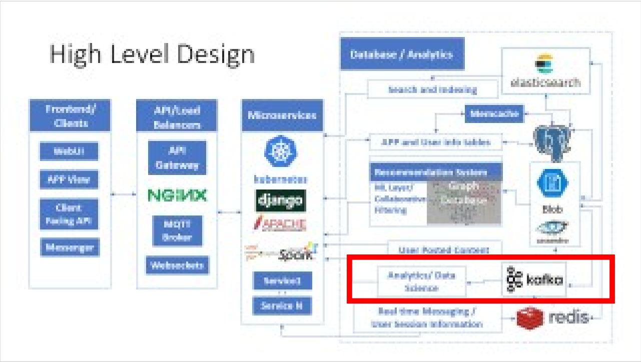
- Generates tailored user feed based on user connections, interests, subscriptions, groups, etc.
- Graph dataset pulls in new data from Cassandra and stores in a relationship topology.
- The relationships are used to curate content, connections and groups recommendations that the user might be interested in.
- These recommendations make the user aware about activity with common interests.
- These will be used to generate user feed.

Search



- Text based search to discover new content based on query.
- Ranks text based on similarity to query.
- Pulls in data from Cassandra and postgresql and makes indexes available to search microservice.
- Elastic search is an ideal choice

Analytics System



- Types of analysis:
  - Predefined insight charts: handled by scheduled async microservices
  - Custom analysis: Data scientist/Analyst can use jupyter interface.
- Produce reports and Realtime insights in a presentable manner.
- Cluster of machines to perform batch analysis.
- Pull data from Redis and Cassandra and make it available for analysis.
- Kafka, Jupyter, Spark and Python ecosystem seems suitable for the requirements.

Thanks