Google Calendar System Design

YouTube - MsDeep Singh

Functional requirements

1. Schedule a meeting
2. Check availability of users
3. Request RSVP
4. Meeting reminders - P1

Non Functional requirements

1. Available
2. Eventual consistency
3. Not a latency critical application of invitee perspective
4. Durable storage - meetings data is removed only if user requested

Capacity Estimations

Assumptions - number of users - 100M

1/10 post invites = 10M

5 posts / user = 50M invites per day

Approx storage for 1 invite =

{Meeting content + userId, invitee list, date}

= 1 KB

Total storage = 1KB * 50M

= 50,000,000 KB = 50GB/day

Storage for 1 year = 50 * 365 = ~20TB/year

High Level Design

Query patterns -

1. Get meetings for a user in Given timestamp
2. Get availability for list of user in given timestamp
3. Check status of meeting - who accepted/declined

Meetings Table

Partition Key - MeetingId (uuid)
Sort Key - userId
Attendees - list{userId: status}
Start Time
End Time
MeetingContent

Schedule Table

PartitionKey - scheduleId (uuid)
Sortkey - userId + meetingId
meetingId
Start Time
End Time

Why Elastic Search ?

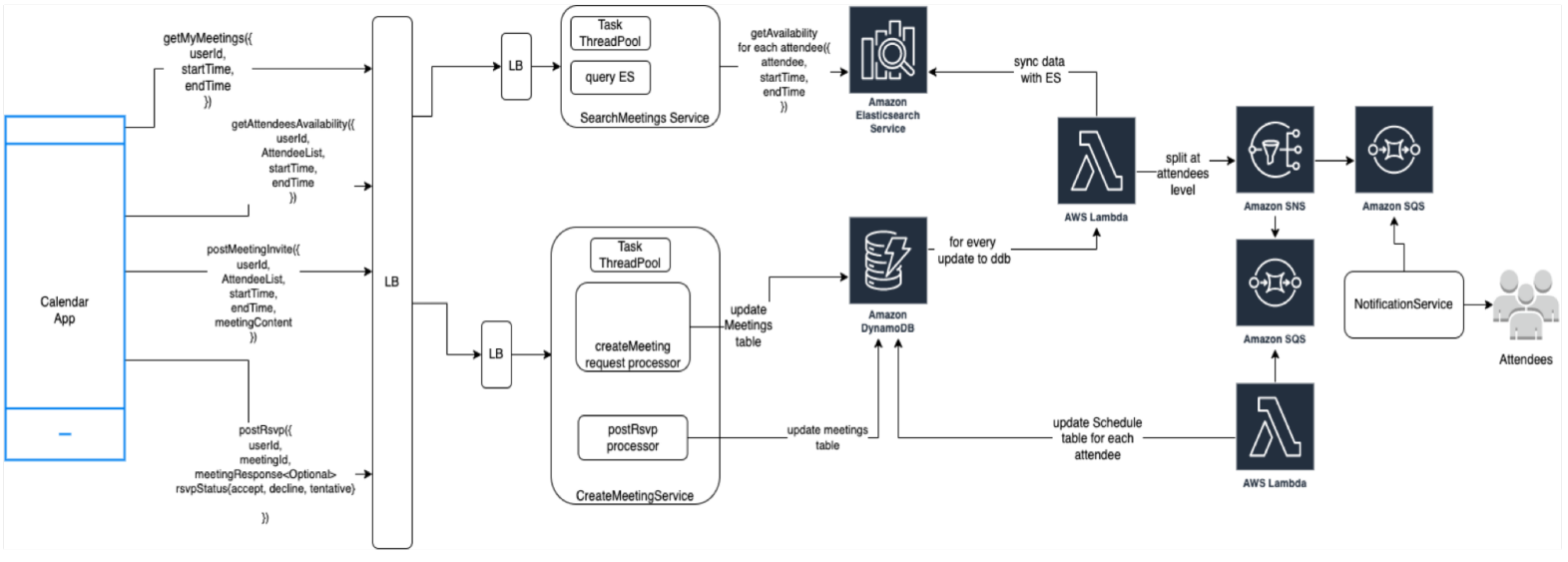
① Make search faster as Dynamo don't have userId as primary key → will require a GSI on top of DDB to build query access pattern.

② ES will help to index every attribute of table

The calendar design can also be achieved without ES with assumption that we don't need search capability on each and every attribute and can be solved via DDB partition key + Sortkey and using GSI(s).

NOTE - Max 5 GSI are allowed on DDB and there will be additional cost associated to GSI.

High Level Component Design →



Points to Consider →

① Is there any special handling required of duplicate meetings? → we can leave this on invitee which meetings to respond / attend.

② Is postMeetingInvite() API complete sync flow?
The API can take significant time - update to DB + update to ES + posting notification to invitees.
→ The sync API can handle till saving to Dynamo and further process can happen in background. API can return ACK once DDB is updated.

③ Why DDB and why not any Relational DB?
→ ACID is not a requirement.
→ There are relationships in table but not to an extent that we infer data combining tables.
→ Priority of availability over strong consistency.

④ API getAttendeesAvailability() can be called in background before postMeetingInvite() API → will help user to check availability of users before sending an invite.

⑤ Handling of Recurring Invites?
As entry in DB is made per user per invite, it will be handled.

⑥ What if the invitee list is too long? will sync API still perform well?
↳ we can think of Hybrid approach if list is too long and causing bottlenecks basis load Testing of system.

⑦ Reminder service → sent remider to user before 30 mins of meeting.

As an extension of current design, this functionality can be offloaded to client Application.

How? API getReminders() is invoked in background every 15 mins (this is randomized, eg 11 mins, 19 mins to avoid Thundering Herd problem) and store the information in APP.

Further APP can display a notification before 30 mins of meeting.

Do share your feedback in comments.

Subscribe Youtube channel for more such content.

Youtube - MSDeep Singh

Instagram - msdeep_14

LinkedIn - msdeep14

GitHub - msdeep14

Happy learning ☺