Kubernette section pretty simple

K8s

1. Kubernettes this is the one here,

Nodes used to run containerized applications,

Control panes run across different applications here

Container

- Used for microservice here, and then

What are some pros?

1. High availability, scability and disaster recovery ehre

2. Has master and worker nodes, kubelet allow each node to talk to each other

3. Each worker has docker container running on it

4.

Docker

1. Make your software/app system agnostic here very important here

2. Container can run mysql db, app and so forth

Note:

This document basically contains short and concise answers to some common questions, but does not contain definition answers, for that pls take a look at the documents in atoms.

What are some important topics

1. Casssadra

2. Database

- types of sharding

3. Kafka

4. Scale redis here

5. System design common questions

6. Scaling websocket (Stateful session)

8.

### 6. What are some good **Failure Recovery and Resilience strategies**

Cassandra section

Key components of Cassandra here

* **Partition Key** - What data determines the partition that the data is on.
* **Partition Size** - How big a partition is in the most extreme case, whether partitions have the capacity to grow indefinitely, etc.
* **Clustering Key** - How the data should be sorted (if at all).

**How to scale Cassandra example?**

How does Cassandra address uneven load between ndoes (Using vNodes)

Since Cassandra use consistent hashing, Cassandra opts to map multiple nodes on the ring to physical nodes in the distributed system.

The nodes on the ring aka : vnodes, are owned by physical nodes. Bigger machines in this would have more vnodes in this case

**How does Cassandra talk to each other?**

Nodes know about each other using the cluster technology

They share cluster information via a protocol called "gossip" (discussed later). Nodes in Cassandra also are able to determine where data lives in the cluster via performing consistent hashing calculations and by knowing the replication strategy / consistency level configured for the data

How does Cassandra work underneath?

1. Use LSM

1. **Commit Log** - This basically is a [**write-ahead-log**](https://en.wikipedia.org/wiki/Write-ahead_logging) to ensure durability of writes for Cassandra nodes.
2. **Memtable** - An in-memory, sorted data structure that stores write data. It is sorted by primary key of each row.
3. **SSTable** - A.k.a. "Sorted String Table." Immutable file on disk containing data that was flushed from a previous Memtable.

How did Cassandra help solve the discord problem?

**Original message table format here:**

CREATE TABLE messages (

channel\_id bigint,

message\_id bigint,

author\_id bigint,

content text,

PRIMARY KEY (channel\_id, message\_id)

) WITH CLUSTERING ORDER BY (message\_id DESC);

What’s wrong with this approach?

The above schema enables Cassandra to service messages for a channel via a single partition.

Partiion key here : channel\_id

But what if some channels have extermly high volumes then? What to do

A new schema is required here

To solve the large partition problem,

1) Discord introduced the concept of a bucket and add it to the partition key part of the Cassandra primary key. A bucket represented 10 days of data, defined by a fixed window aligned to Discord's self-defined DISCORD\_EPOCH of January 1, 2015.

2) The messages of even the most busy Discord channels over 10 days would certainly fit in a partition in Cassandra. This also solved the issue of partitions growing monotonically; over time, a new partition would be introduced because a new bucket would be created.

3) Finally, Discord could query a single partition to service writes most of the time, because the most recent messages of a channel would usually be in one bucket. The only time they weren't is when 1) a new bucket was created based on time passing, or 2) for inactive Discords, which were the significant minority of queries to the messages Cassandra table.

A new table is then created below:

CREATE TABLE messages (

channel\_id bigint,

bucket int,

message\_id bigint,

author\_id bigint,

content text,

PRIMARY KEY ((channel\_id, bucket), message\_id)

) WITH CLUSTERING ORDER BY (message\_id DESC);

Cache section

Different types of caching mechanism

Write thru cache

Under this scheme, data is written into the cache and the corresponding database simultaneously, completely same consistency

Database section

How do we deal with hotspots?

Can we use full text index?

**What are file storage options?**

**Usually blob storage** here, Amazon S3, can also use CDN with this as well, to serve static image, distrbiute image across the globe so it can be faster for storage

How to make sure transaction is carried out correctly when dealing with multiple dbs in a microservice environment?

1. Using saga or 2 phase commit, 2 PC commit is usually better anyways

What are some top sharding techniques?

Range based sharding:

Hash-based sharding

**Why is hash-based sharding good?**

1. It uses consistent hashing,

2. One of the main advantages of hash-based sharding is its ability to distribute keys fairly among the shards. By applying a hash function to the keys, the technique helps mitigate the risk of hot spots.

However,

1. Hotspot issue is not removed here, just reduced by a bit

2. Can not perform efficient query

How are shards dynamically added?

1. Automatic scaling:

- Automatically create or remove shards based on data load.

Dynamic sharding is typically implemented using partitioned tables that can be dynamically split or merged. This can be done with scripts or automated tools that monitor data distribution and adjust shard configurations accordingly.

Why is database migration done?

What problem are we trying to solve?

Keep this in a source of truth here

-

- Sql database, sql workbench here.

Why is this actually done?

What does high cardinality key?

It means a key with value that has a high number of values,

What are the 3 top choices for database?

1. Mysql, sql
2. Nosql : couchbase, Mongodb
3. Ever increasing nosql (Columnar DB)   
   Cassandra Hbase

How to achieve high consistency in database?

1. For systems with multiple nodes, utilize distributed transactions to ensure all related updates are committed together, maintaining consistency even during network failures. Done using 2 phase commit as said before

What are the 3 types of consistency? Can you give examples and let me know which one is suited for your particular database?

Strong consistency, eventual consistency and weak consistency

How does nosql query data for multiple tables?

* Embedded documents (a document within a document)

What are the 2 types of locking

1. Optimisitc and pessimistic

Optimistic is used with version here

Can look at hotel booking system for the applicatino of this?

How does full text search like Elastic search gets integrated with postgresql?

- an example can be found in the ticketMaster example, we must keep our elastic search in sync with the database here

-

Elastic seach section

1. Inverted index

What’s another alternative to elastic search?

1. We can use inverted index here

**Kafka section**

**shoudl you have producer and consumer on the same service kafka?**

**Yes,** you can definitely do that, this can be seen in this example above.

**https://dzone.com/articles/kafka-producer-and-consumer-example**

How do we scale kafka event-driven microservice here?

How to increase # of partition in a group?

What’s the pull approach from kafka?

This means service is continuously pulling from the pub/sub in kafka.

What’s a partiion and how does it help with parallel processing?

1. How to scale kafka

## Scaling Kafka Cluster Horizontally

Scaling a Kafka cluster horizontally involves adding more brokers to the cluster to distribute the workload and handle increased data throughput. The process typically includes repartitioning existing topics, adjusting topic configurations for new partitions, and closely monitoring and managing the cluster after scaling. Let's go through each step in more detail:

1. **Adding more brokers to a Kafka cluster:**
   * Provision new servers or instances that meet the hardware requirements for Kafka brokers.
   * Install Kafka software on the new machines and ensure they can communicate with the existing **Kafka cluster and ZooKeeper ensemble**.
   * Update the configuration files of the new brokers to match the existing cluster's configuration.
   * Start the new Kafka brokers, and they will automatically join the existing cluster as part of the new Kafka broker group.
2. **Repartitioning existing topics:**
   * Before adding new brokers, determine the optimal number of partitions for each topic. More partitions allow for better parallelism and improved throughput.
   * Use Kafka's partition reassignment tool to rebalance partitions across the new and existing brokers, ensuring an even distribution of partitions.
   * The reassignment process should be performed carefully to avoid unnecessary data movement and maintain the order of messages within each partition.
3. **Adjusting topic configurations for new partitions:**
   * After repartitioning, update the topic configurations to reflect the changes in the number of partitions.
   * Review the replication factor setting to ensure that there are enough replicas to maintain data redundancy and fault tolerance.
4. **Monitoring and managing the cluster after scaling:**
   * Regularly monitor the performance metrics of the Kafka cluster, such as throughput, latency, and resource utilization, using monitoring tools like **Prometheus, Grafana**, or other Kafka-specific monitoring solutions.
   * Check the consumer lag to ensure that consumers are keeping up with the data flow and not falling behind.
   * Monitor the disk usage on brokers to ensure that there is sufficient storage capacity for data retention.
   * Keep an eye on the health of ZooKeeper ensemble as it is critical for Kafka's metadata management.
   * If needed, adjust resource allocations for brokers based on the observed metrics to ensure optimal performance.

s

1. How do we make sure no duplicate in kafka producer?

[Idempotent](https://en.wikipedia.org/wiki/Idempotence) producer can help resolve the issue. To achieve this, the order service (if the order service is the one publishing) should be assigned a unique producer ID (PID) and each message published should be given a sequence number. The combination of PID and the sequence number is tracked by Kafka as a unique ID for a message. So, when a retry happens for an existing message, Kafka gives an acknowledgement back without appending the message to the log.

2. How can we make sure no duplicate on consumer?

- Message id, store that in the database, always check before stroing the message in the database as said

**Redis section**

1. How do we configure the data in redis with a ttl

Check ticket master example in the answers sheet

2. How do we scale redis?

There is the fixed approach and the hashed approach (where we are merging k top score lists) here

**Why is consistent hashing so important?**

It’s the technique to achieve even distribution of data in horizontal scaling across servers or database servers, this is why this is important.

What is it basically?

* Map servers and keys on to the ring using a uniformly distributed hash function.
* To find out which server a key is mapped to, go clockwise from the key position until the first server on the ring is found.
* When new server is added, only one key needs to be remapped here

**If we have a non-unifomr key distribution on the ring, how do we solve this problem?**

This is done using vNodes on each server node

**General microservice section**

**What’s a stateful service?**

A stateful service is one that manages connection, like with weboscket

What does the circuit breaker pattern look like?

How to achieve a low latency here?

1. Index, using caching, load balancing

How to achieve high availability

- using load balancing, using redundancy and replication, failover clustering for database here

- Health monitoring and alerts here

-

How to decide which shard to call?

1. We can use consistent hashing here

cacheHostNumber = hash\_function(key) % numberOfCacheHosts

What’s the pros and cons of using a cron job?

Scalability issues in large systems:

Managing cron jobs across a large distributed system can become cumbersome due to the centralized nature of cron.

**What are some alternatives to using a cron job?**

A distributed job scheduler

**How Slack Built a Distributed Cron Execution System for Scale**

What’s apache spark?

Used for stream processing with apache kafka

**What are some real use cases for this?**

Big data, data warehousing and etc.

Kafka section (Ad click aggregator)

1. How do we make sure that there is no duplicated data when delivering from kafka?

Redis section

1. How do we scale redis?

How do we know which shard to write to? Take a look at gaming leaderboard problem x

**Websocket section**

What are some challenges?

Using spring database here and what are some of the good things you can try?

What are some basic components?

1. We need a topic/channel for updating msges here

WebSocketConfig.class

We need topic/cricket, topic/orders

The above defined in websocketConfig class,

2. A notificationController

Can send to the actual broker here

@sendTo(“topic/notification”)

Every client connection subscrbing to this topic will then listen to it and then

Print out the node you are listening to

3. And then open a few tabs

1. May be hot spots as you scale here

- Too much load may get directed to a few instances, leading to uneven resource usage and potential bottlenecks.

2. Connection management

|  |  |
| --- | --- |
| WebSocket doesn’t automatically reconnect if the connection drops | Requires manual reconnection management, such as heartbeats and pings, need to manage potential data loss during disconnections and pay attention to factors such as message ordering |

What’s a sticky session?

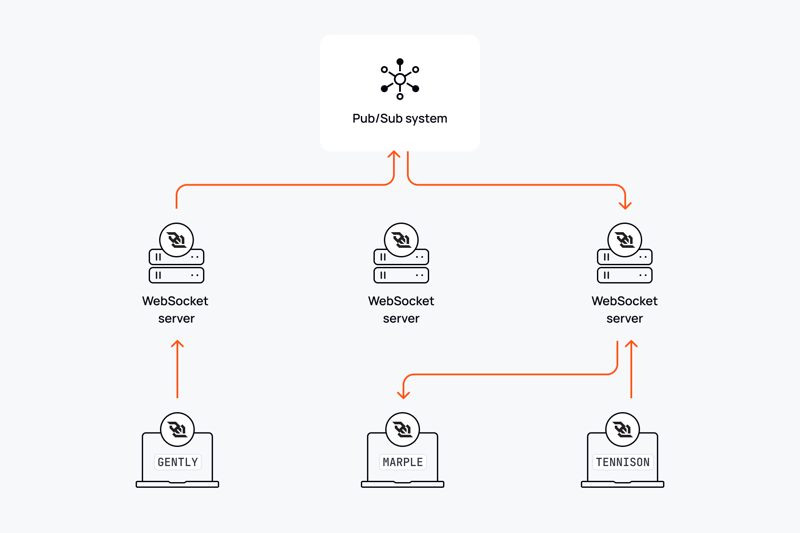
One way is to make sure that specific clients connect to the same server every time they reconnect. These are sticky sessions.

Here’s how it works: once a client establishes a connection with a server, any subsequent reconnections are directed to the same server. This avoids the need to synchronize session state across multiple servers. While this helps maintain session continuity, it also introduces some challenges. The load balancer needs to track which client belongs to which server, which adds complexity and can make that load balancer into a single point of failure.

The load balancer lets us know which client connects to which server

Why is pub/sub important in a chat app here?

For example, in a realtime chat app, rather than having every WebSocket server manage both connections and message routing, the servers handle the connections and leave the pub/sub system to broadcast messages to all clients subscribed to a chat room or channel. By offloading this responsibility, pub/sub allows your architecture to scale without requiring every server to stay perfectly synchronized.



In the above,

Gently publish a message and then the Marpole and Tension get that message then.

In a chat service, how do we know which user connects with which service?

We have a service discovery that takes care of that here, once user logs in, we decide on this here.

I think using a pub/sub is better here.

**In a chat service, how do we know userA sends a message to userB if they are on different server?**

Option 1: Websocket manager

The by using a central "message broker" or "WebSocket manager" that keeps track of which server each user is connected to, allowing the message to be routed to the correct server where the recipient is currently logged in; if the recipient is offline, the message is typically stored in a database until they come online again.

 Option 2: A pub/sub for this here:

Pub/sub

1.  A publish-subscribe pattern can be used where the sending server "publishes" the message to a topic associated with the recipient, and the recipient's server "subscribes" to that topic to receive the message.

- this ensures this is async as said before this is very important here.

Rely on a websocket manager here.

1. How does server sent event work in sprint?

Webflux

2. What are the disadvantages and advantages of using server sent event in spring?

**Case studies**

Uber

How do we prevent multiple ride requests from being sent to the same driver simultaneously? (Uber)

1. Lock with TTL with in-memory database using Redis here

The Ride Matching Service attempts to acquire a lock on the driverId in Redis. If the lock is successfully acquired, it means no other service instance can send a ride request to the same driver until the lock expires or is released.

**Ad click aggregator example**

How do we make sure that advertisers can query metrics at low latency? (how many times an ad is clicked for a time window frame

* Using stream processing with Apache spark
* Using batch processing here

**Url shortener**

How to deal with having a centralized counter? (Url shortener)

Using redis to store a single counter that can be accessed by everything

**Hotel reservation section**

**How to prevent the 2 users from taking 1 transaction?(Can check the hotel reservation system)**

1. Pessimistic locking
2. Optimistic locking (using version)

**Chat system**

How do we make sure each client will get a message when sender sends message?

Each client will be subscribed to a particular pub/sub and the pub/sub is connected to a websocket server and when things happen it will comes to th

**Ticketmaster case study:**

**How to ensure that the ticket is locked for the user whilie they are checking out?**

1. Use distributed locking with redis ttl here

**How do we make sure that the booking map is always up to date?**

1. We can use SSE for real time seat update

2. using virtual waiting q for popular events for websocket connection

1. **How can you improve text search to ensure we meet our low latency requirements?**

1. Create indexes on event table,

2. Use full text indexes in the database such as mysql or postgresql here:

**How does elastic search sync with Postgresql?**

we can use change data capture (CDC) for real-time or near-real-time data synchronization from PostgreSQL to Elasticsearch. This setup captures changes in the PostgreSQL database, such as inserts, updates, and deletes, and replicates them to the Elasticsearch index.

Building a location service

1

Url shortener

1. How to make sure each url is unique?

Use global redis counter and a base 62 hashing here

2. How to make sure redirects are fast?

Use a mapping between short urls and long urls here

How would users be able to saerch for words by keyword?  
  
Why using the like keyword is a bad approach?

The naive solution to this problem is to keep all of the posts in a relational database and use a query like SELECT \* FROM Posts LIKE '%$keyword%';. This would technically return the correct results for a given query!

The above is a very bad solutino because it’s slow ehre

Using inverted index in elastic search optoin

1. An inverted index is a data structure used to store mapping from content, such as words or numbers, to its locations in a database, or in this case, documents.

Suppose we want to find all the words that contain the word “great”, how do we do that?

A screenshot of a computer screen

Description automatically generated

The above is the solution you are looking for here

1.

Case study here:

How would Users be able to get search results sorted by recency or like count?

Using multiple indexes here

A different approach would be to have two separate indexes: one sorted by the creation time and one sorted by the like count (I'll refer to these as the creation index and likes index going forward).

One for the creation index and like index here, these 2 are very important as said.

Using our Redis-based approach from earlier, we can have separate keys depending on whether we're sorting by Likes or Creation date.

For the creation index keys, we can use a standard Redis list. We're always going to appending to this list and our queries will only be taking from the last elements.

For the likes index, each key can use a [**Redis sorted set**](https://www.hellointerview.com/learn/system-design/deep-dives/redis#redis-for-leaderboards). The sorted set allows us to keep a list of items ordered by a score in the same way that a priority queue or sorted list might work, with the same time complexity of insertions and queries.

When a new post is created, we'll add it to both indexes for every keyword it contains. When a like event happens, we'll update the score in our sorted set for the likes index.

1) How can we handle the large volume of requests from users?

1. Using redis

2. Using CDN cache here

Our in-memory reverse-index based system is quite fast, but we're going to be handling a lot of traffic. We had some convenient requirements earlier that might make our job even easier. Two requirements in particular:

1. We do not have personalization, so if you and I are searching for the same thing with the same parameters, we should get the same results!
2. We have up to 1 minute before a post needs to appear in the search results.

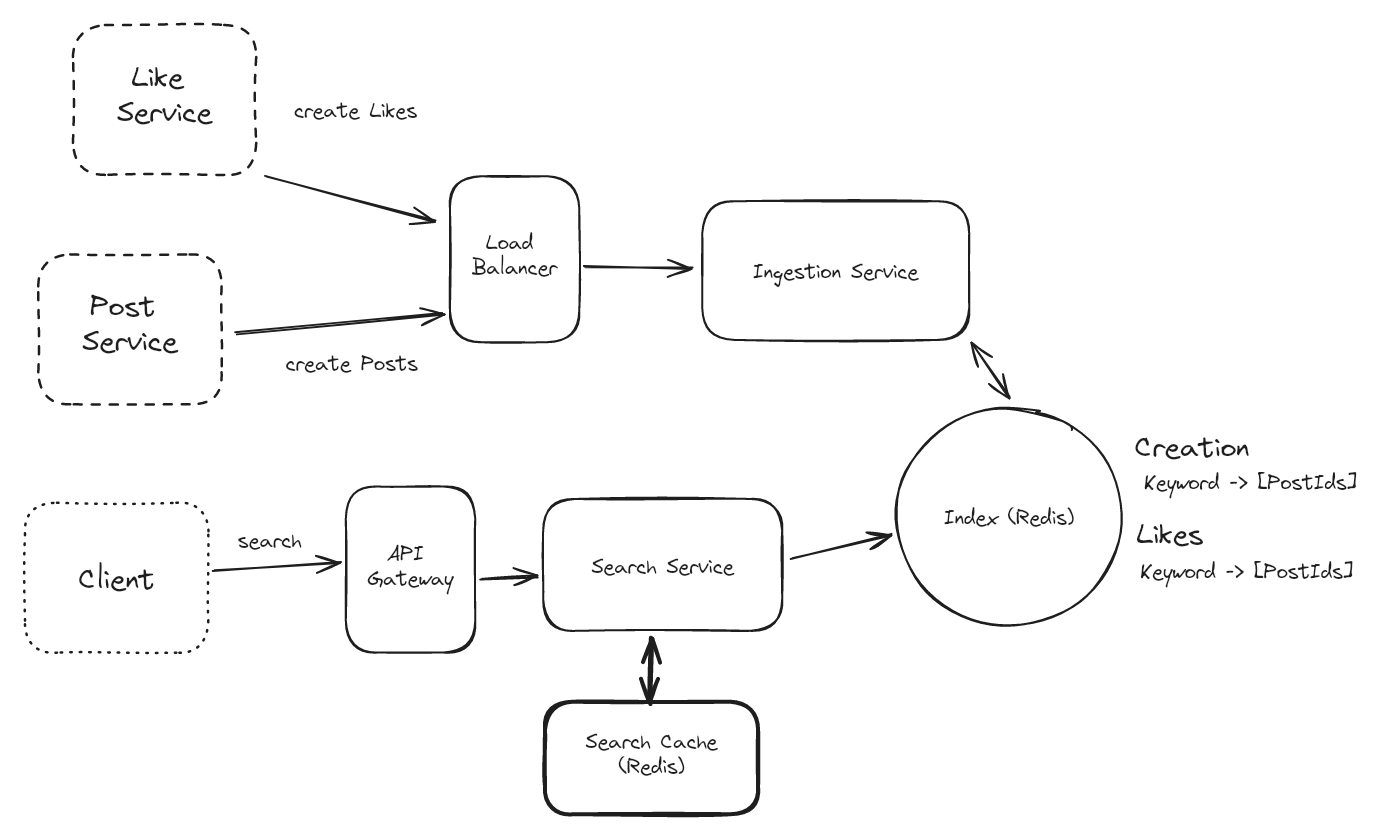
Caching sticks out here as the obvious tool for us! Any time we can tolerate stale data and we have duplicate requests coming through, we should consider whether caching is appropriate.

**Good Solution: Use a distributed cache alongside our search service**

Approach

One option for us is to add a distributed cache alongside our search service. This cache would be responsible for storing the most recent results for a given search query. When a search is performed, the service would first check the cache to see if the results are available. If they are, the service would return the results from the cache. If they are not, the service would perform a full search and store the results in the cache for future requests.

We'll want to put an eviction policy on our cache to ensure stale results don't stick around. Since we have up to 1 minute SLA on new posts, we can institute a TTL of < 1 minute on our cache. This will guarantee we're never serving results that might not contain newly created posts.



Search Cache

**Great Solution: Use a CDN to cache at the edge**

**Approach**

Use CDN

How do we handle the large volume of writes for post?

1. Using kafka is step 1, fan out the writes to muliiptle instances here

**What about for the like event?**

1. One approach we can take is to batch the writes for likes. Instead of writing every like update to our indexes, we can batch likes for a given postId over a period (like 30 seconds). Then, instead of needing to write 500 times for a particularly viral post, we can make 1 update with an increment of 500.

To do this we’ll need a secondary "batcher" service which accepts like events and aggregates them over a fixed window (maybe 30 seconds) before writing them back to Kafka to be consumed by the ingestion service.

**How to optimize the storage with our current system?**

Like how do we deal with too many indexes?

Use batch job to remove them when they are no longer needed here.

we can run a batch job to move rarely used keywords to a less frequently accessed but ultimately cheaper storage. One way to do this is to move these keyword indexes to cold, blob storage like S3 or R2.

On a regular basis we'll determine which keywords were rarely (or not at all) accessed in the past month. We'll move these indexes from our in-memory Redis instance to a blob in our blob storage. When the index needs to be queried, we'll first try to query Redis. If we don't get our keyword there, we can query the index from our blob storage with a small latency penalty.