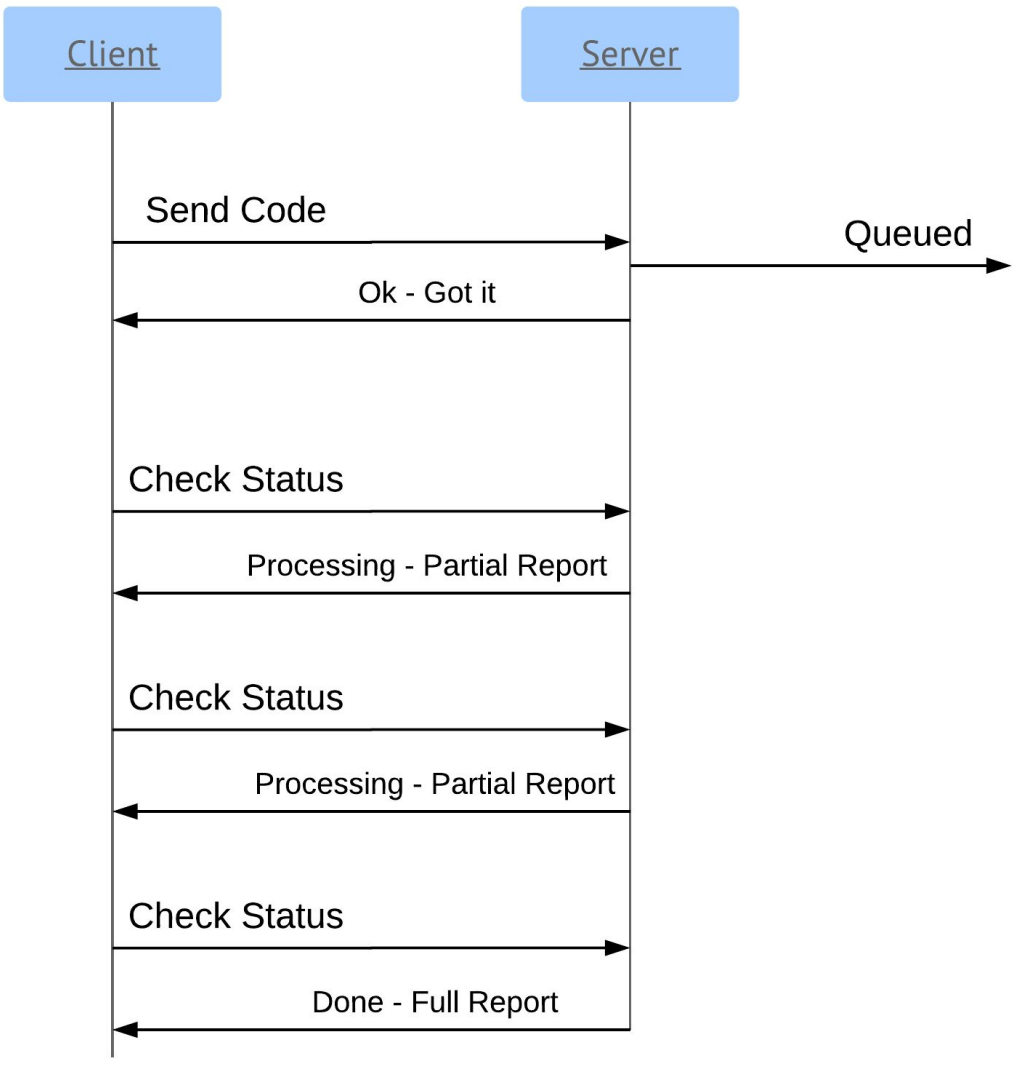
CLIENT - FLOW

1. Fetches the question, question metadata (time counter, minimum time taken for the code processing to finish, etc) and input sample to show
2. Submit/Test ->
    a. Sends the question id and the code to server
    b. Gets a 200 response when uploading is finished
3. Status ->
    a. Polls for status after minimum time taken for code processing
    b. After that, Short Polling every 2-3 seconds
    c. Status will include all data test cases passed, score, etc



SERVER - LOGIC

SEND CODE

1. Receives the question id, user token and code as a POST request
2. Creates an unique key for question and user, something like <user_id>-<question_id>
3. **Enqueue in messaging queue question id, unique key and the code**
4. Save in cache the status UNIQUE_KEY => ENQUEUED, EMPTY REPORT, if not already present or status not PROCESSED
5. Save in cache the status UNIQUE_KEY => REENQUEUED, EMPTY REPORT, if already present and status not PROCESSING

CHECK STATUS

1. Creates an unique key for question and user, something like <user_id>-<question_id>
2. Check in cache
3. If not PROCESSED, return the status and report in the cache, if not PROCESSED
4. If PROCESSED
    a. Save attempt stats in DB with a new attempt id and user_id, question_id, successful_testcase_count code blob storage url
    b. Save attempt testcase stats in DB attempt_id, testcase_number, status for analytics purpose
    c. Set the status value as DONE and expiration of the record in cache for 5 minutes
    d. Return the status and report in the cache
5. If not in the cache, fetch from the database

QUEUE CONSUMER

1. Create a directory with the unique key in the file system
2. Fetch questions details and language details from cache or fetch from database and cache
3. Create a file of the code using the name in question metadata and type in language metadata
4. Get script to compile from language metadata
5. Get script to run the code with test case as parameter
6. Run compile code
7. Update the status in cache as PROCESSING
8. Iterate on Test cases
    a. Check status of unique key in cache
    b. Exit if ENQUEUED
    c. Run the code with the test case
    d. Fetch the logs and add in test case report
    e. Update the report in cache
9. Update the status in cache as PROCESSED
10. Move the code to Blob Storage
11. Save blog storage url in cache
12. Delete the file
13. If there is any way that we do not need to save the file in filesystem, compile the file in-memory and run it in-memory, then the time of saving to secondary memory is saved

SERVER - STORAGE/BANDWIDTH ESTIMATION

ASSUMPTIONS

One User attempts max average of 50 questions in a lifetime
One Question has max average 20 test cases
User does max 2 questions in an hour
User make 5 submissions per question = 10 submissions per hour
Time to process one submission = 1 second
Concurrent Users = 2 M
Users Per Hour = 3 M
Submissions Per Hour = 3 M * 2 questions * 5 attempts = 30 M

DATABASE

<u>One Record Size</u>
1 User = 5 KB of metadata + 1 KB of extra stats = 6 KB
1 Language = 2 KB max of total scripts + 2 KB of extra metadata = 4 KB
1 Question = 15 KB for text + 1 KB test data open + 1 KB metadata + 1 KB of custom scripts if required = 18 KB
1 Test Case = 4 B of question_id + 1 KB input + 0.5 KB expected output = 1.5 KB
1 Attempt = 4 B of question_id + 4 B of user_id + 1 B of successful_testcase_count + 256B url = 10 B approximately
1 Testcase Attempt = 4 B of attempt_id + 1 B testcase_number + 2/3 b status = 5 B max
1 Report = 20 KB
1 Code = 20 KB

<u>Record Count</u>
User Count = 5 M
Language Count = 20
Question Count = 100 K
Test Case Count = 2 M
Attempt Count = 250 M
Test Case Attempt Count = 5 Billion

Total Database Size = 30 GB + 80 KB + 1.8 GB + 3 GB + 2.5 GB + 25 GB = less than 60 GB

BLOB

Code Count = Attempt Count = 5 M
Total Blob Size = 100 GB

Possibilities - Question and Test Cases can be put into Blob Storage, but then we have to manage a lot of ids

BANDWIDTH

<u>One Request Size</u>
Code Send Request = 20 KB of code + 1 KB of user token + 4 B of question id + 1 KB metadata = 22 KB
Code Send Response = 1 KB status
Check Status Request = 1 KB
Check Status Response = 1 status KB + 20 KB report = 21 KB
One Submission Requirement = 1 code send + 4 check status = 23 KB + 4 * 22 KB = 111 KB

Total Bandwidth Requirement Per Hour = 30 M * 111 KB = 3330 GB = 3.3 TB
Total Bandwidth Requirement Per Second = 3.3 TB / 3600 = 1 GB (upper round) = 2 GB (even upper to be on safe side during drives)

CACHE

Average Per User Cache Size = 21 KB report response of one question + 21 KB report of previous with probability (cache expiry / time for one submission) ⅛ = 24.5 KB ~ 25 KB
Total Cache Size = 2 M * 25 KB = 50 GB + 1 GB for caching questions, etc = 51 GB

QUEUE

<u>Count</u>
Submissions Per Hour = 30 M
One consumer process submissions per hour = 3600
Minimum Consumers required = 30 M / 3600 = 8333 ~ **10000**
Codes in queue = 2 M

Queue Size = 10 % submit together = 2 M * 10% * 21 KB = 4.2 GB

SERVER - COMPONENT

DATABASE

1. Document Based NoSQL like Couchbase/MongoDB/DynamoDB/DocumentDB
    a. User
    b. Language
    c. Question with test case in the document
2. Wide Column Store like Cassandra for data which are write extensive
    a. Attempt with test case results as frozen list in it
3. Relational
    a. Nothing as such that would prefer RDBMS over NoSQL in this case
    b. There might be in future

CACHE

1. Redis Cluster - Redis mainly because it has sorted set
    a. Attempt Report
    b. User Ranking by scores
    c. Question Ranking by attempts
    d. etc

BLOB

1. S3
    a. Code (Can be 2 for each user and each question, last attempt and best attempt, rest can be deleted)
    b. Logs (Optional as the logs can be regenerated from the code)

QUEUE

1. KAFKA/SQS/RabbitMQ
    a. Code

QUEUE CONSUMER

1. **AWS LAMDBA** - 512 MB SSD and 128 MB RAM is enough for one code to test
    a. By default maximum concurrent execution is 1000
    b. Will need to get it upgraded as the load reaches peak

ANALYTICS SERVICE

1. Storm/Spark
    a. Any realtime processing or reporting required
    b. For future use

LOGGING SERVICE

1. ELK Stack
    a. Logs from Web App
    b. Logs from Queue Consumer

SEARCH SERVICE

1. Elastic Search (Can use same as for Logging Service)
    a. Questions

HackerEarth Architecture

Not mentioning any specific tech stack name, because different for AWS, GCP, AZURE, etc