**System Design**

Many systems design questions are intentionally left very vague and are literally given in the form of “Design a white dragon”. It's our job to ask clarifying questions to better understand the system that we have to build.

I give an example of some of questions i would ask if i asked to design a system like the StackOverflow website and their answers are.

1. Are we designing all the system or a part of it ?

Answer: “The new home page new user/profile side and login and the questions”

I consider the core user flow includes:

a- Home page,

b - Search Text Based (maybe Elasticsearch fits)

c - List of questions,

d - Response to questions,

e - Mark answers as complete/positive or negative.

2. Our new “StackOverflow” system seems like a system of criticality but not like a hospital system or an airoplane system so the availability is important but not life or death critical; Are we okay with 3 to 4 nines high availability (8 hours to 4 per year) i think.

We can use multiple Load balancers probably 2. They should be configured in such a way that they can handle the sessions. Maybe by sending the same ip to the same backend every time. Or store them inside a database, or some shared memory.

We can have some **DNS load balancing** to route the request to the cluster closest to the user. (number or requests > 800 HTTP requests a second that the service already has) and within a region we can have some **path-based load balancer if there are not similar services –** The main platform, API, Payments Service, UI (text based searching with questions, jobs posting section, chat) We can do some **round-robin load balancing** for the API and the searching which included in path-based-load balancer.

3. How many customer should we be building his for .Global scale, to some countries or to one country?

The new Stackoverlfow will have global scale with 100.000 thousands to 1 or 2 million

users. Every month > 100mil page views and everyday > 5 mil. Top users come from USA and Asia.

4. Do changes happen often ?

“Every couple of months and the changes should arrive to the user within an hour globally.

Step1.

User data like the UI, or the last seen questions or frequent question seen by a user. These drives us to the need of two layers of caching on client side which will be on the clients’ browser.

Step 2.

VarnishCache is a reversed proxy caching, it sits between the Server and the Internet so the workload to the Server caching will get reduced and the speed will go up further. Also it is free.

Step 3

On the server side we can cache the solved questions with its answers and the questions with positive votes more than 4. We assume that these questions were useful for other users and future users will find them useful as well. As such we do not cache every single question.

5. How much of the searching speed, questions ranking.

I think the searching speed then text based searching are most important in the UI so we can use Elasticsearch, it is a NoSQL DB. It offers the speed and the text based searching.

6. While latency seem important also the data capacity seems quite huge. A) Are we Okay storing all the questions on data centers or we clean them often?. B) We can use distributed datacenters based on the most throughput ?

We will use CDN (content deliver network) referred to geographical groups of users This will speed up the JS, HTML and the UX in general. The CDN also prevents DDOS attacks. Also it is okay to delete questions with negative feedback more than -3 points or if marked as duplicated.

1. We will use 2 layers caching as we proposed in the previous question and the storage will get managed better.]
2. Its is fine to distribute the most of the resource to USA, then to Asia, then to EU,

using GCP, Azure Cloud with MS SQL Server or Oracle Cloud with Oracle RDBMS or AWS.

7. Access Control

Users with no log in or no registration or users with -15 points cannot alter or edit the content of the thread’s question, they are in read only mode. We can do this by checking his profile status and if he logged in.

8. User Data Storage and DB Schema.

Since as we discussed the decentralized datacenters and the data will queried a lot I believe a SQL database like MySQL, PostgreSQL or MS SQL Server(with LINQ for Object modeler), Oracle with (Django or other ORM) are the choices with the last 2 as the main candidates.

Also noSQL is more appropriate for fast querying so ElasticSearch or Clickhouse or MongoDb for the searching.

Top RDBMS Tables:

user\_registration (uid, username, password, age, sex, nickname, job, company)

user\_login (uid, username, password, tags)

users\_questions(uid, question\_id, tags)

--Searching and questions are the main load and they can be a text based db storage.  
It Is okay to be a new NoSQL database. Use of data of RDBMS to Elasticstack through Filebeats, Logstash, Elasticsearch.

questions(fk: uid, question\_id, votes, title, text, tags, solution\_status, datetime)

searching(question\_id, title, text, tags)

Elastic Stack will gather these two tables from the RDBMS and create multiple indices from their combination.

9. Monitoring and Logging

We can have monitoring to see the average response time of users, the number of events and questions we process each day.

Also logging to help with bugs, user’s undesired actions or malicious attacks.

Monitoring and logging will help us to scale easier with more CPUs, memory, more security from DoS attacks if the needs require it in the future.