**System Design (Scalability and Operational Performance)**

**What is System Design?**

Amazon SDEIIs design software solutions to enable new features or improve their team's software. SDEIIs must solve difficult problems arising from visible risks or roadblocks. They make technologically appropriate decisions in the context of the broader business and technology strategy they work within. Their design decisions consider load and usage patterns and future needs.

SDE IIs focus on the end-to-end aspect of the design. The interviewer’s expectations are for you to understand the components and come to a decision on which components to use. For example, if you the interviewer asks you to “Tell me how you would design a service.” The interviewer might ask to design this service with two APIs, one is a getter and the other is a setter. For the getter, the interviewer is expecting you to think about caching layers. For the setter, the interviewer wants to hear about the database and maybe how do you multiple times write it to the same database. Consider the database going to some schema design if needed. It’s all end-to-end.

Prior to drawing out the design, make sure to ask clarifying questions to disambiguate the problem. Write down a list of requirements needed for the problem and talk it through with the interviewer. Even if you think you understand the requirements, please still make sure you talk it through with your interviewer before you start the design. This session will in total run about 30 minutes.

There are five elements we consider for the system design question:

1. **Technology related:** the principles of our system design questions can apply in all sorts of context that aren't technology.  Nevertheless, the best evaluation and candidate experience come from staying closer to the job relevant skills.  Our interviewers will not ask candidates to design a system that would not be in scope for the SDE job family.
2. **Create not describe:**  We're not trying to get the candidate to tell us about an existing system or pattern they've seen somewhere.  We're testing whether they can design systems that will solve real problems (and not create them).
3. **High level:**  Our interviewers are looking for questions that analyze systems.  Not minor details and certainly not coding since we have other interviewers covering that skill.
4. **Ambiguous:**  Dealing with ambiguity is one of the most important of our functional skills. Leaving room to find out if a candidate can navigate that is important in a system design question.
5. **Something the interviewer knows:**  If the question is outside of your area of expertise, you can't adequately evaluate the answers. You will struggle with follow-up questions.
   1. But beware of the bias that comes from having extensive experience (or from having asked the same question a 100 times and having observed every permutation of the answer).
6. **Tailored:**  Ideally the interviewers are not asking a question about Android App design to a candidate who has 30 years building hyperfast systems for trading.  And ideally the interviewer isn’t doing the reverse of that either.

**How to Pass System Design**

\*In the chart below, there is no absolute scoring on the criteria and they are equivalently important. It is about making decisions based on trade-offs and risks seen in the interview (and obviously the more mild strength/full strength data points the better).

|  |  |  |
| --- | --- | --- |
| **Criteria** | **Strength** | **Example** |
| **Ambiguity & Requirements** | * Asks key questions to understand the problem and requirements. * Assumptions - if made - are reasonable and clearly stated. | If there is a web UI, relevant questions may include:   * Who are the users * How many of them are there? Concurrently? * Is it in the internet? * Does it need authentication/authorization. |
| **Comprehensive Solution** | * Makes sure to understand the entire problem before diving deep. * Picks top-down or bottom-up approach as appropriate, and sticks with it. * Proposes appropriate solutions/components/technologies, and can explain why they are appropriate. * The candidate may not be familiar with specific technologies, but can still describe what they want. Saying "I'd like a HashMap wrapped in a service" is a fine substitute for "KV store" or "DynamoDB". | If the problem is well suited for a top-down approach, a great way to start would be with very few, very high-level building blocks (e.g. "Ingestion", "Processing", "Presentation") before further refining what those blocks do, before discussing table schemas and protocol-level details. After you are content that the candidate has a firm grasp on the overall problem, choose areas for them to explore in more depth.  Question buzzwords. If a candidate proposes to solve poor website performance with caching, make sure they considered whether data is suitable for caching in the first place. What happens if entries get evicted? If a candidate proposes to use noSQL to make the data store scale, ask them to articulate what, exactly, causes bottlenecks in a RDBMS for this particular use case. |
| **Abstractions and Boundaries** | * Components have clear boundaries * Interactions between the components are well defined and easy to explain | * Imagine Kindle. Can you clearly identify what's the responsibility of the Server, and what should run on the client? * Imagine you have a UI accessed by a billion people on the internet and another UI for your internal admins. Both could be hosted on the same webserver. But should they? * Imagine you need to store some data. Does all of it belong in the same database, or even the same data store? |
| **Scaling** | * Able to explain which parts of the design are likely to become bottlenecks at scale * Can use order-of-magnitude to quantify those limits, and discuss potential solutions | * If you need to perform 20 API calls in a sequence, did you recognize that ping time is non-zero, so the whole sequence will be expensive, and you can't just add servers to make it better? * How about trying to put a large volume of data in an RDBMS - where will this break, and why? * And if the proposal is noSQL, then what flavor (KV, doc, graph store, etc.) and why is it the best choice? |
| **Staying on the rails** | Keeps in mind the functional requirements, and ensures that the proposed solution satisfies them | Table scans, even on large data sets can be perfectly fine. But if your requirements state you need to present the results in a UI, table-scanning 10^9 rows is probably a dead end |
| **Considering Alternatives** | * Is able to articulate pros and cons of various viable approaches. * If you are unsure whether they considered alternatives, ask. They may have missed them, or dismissed based on extensive prior experience. * Is able to commit to a chosen approach * Prefers two-way door solutions if available and appropriate. | When choosing whether to push vs pull data, the candidate should be able to think about how to handle outages, how to scale the receiving end, whether the sender or the receiver have a better idea when to initiate the transfer, and so on. |
| **Avoidable Mistakes** | * Not only presents correct solutions, but is able to think a few steps ahead and look around corners * Avoids rework due to foreseeable mistakes.   To be clear, iterative approach is fine, making mistakes is fine, but making mistakes that should've been obviously foreseeable indicates lack of experience. Interviewer needs to exercise judgement on what should be easy to anticipate for a non-Amazonian | The candidate proposes storing a large amount of data in RDBMS, recognizes ingestion will be a bottleneck, switches to a sharded KV store, just to realize he won't be able to pull the data efficiently enough to satisfy tight UI latency requirements, and finally throws a cache in there to make it all work. |
| **Data-Intensive Scenario** | Uses appropriate data stores. Understands strengths and - more importantly - weaknesses of different data stores. Can dive deeper into schema design or sharding strategy if needed | Saying "I'll use non-relational database because it's faster than SQL" is not enough.   * Why is RDBMS slow for this particular case? * Which specific non-relational tore (KV, document store, graph store, etc.) would work better? * Why, specifically? * Is it because transactions are a natural bottleneck, and we can - say - sacrifice 'A' in ACID? * Is it because you can use eventual consistency? * If it is because there's sharding, explain the sharding strategy |
| **API Design** | * Comes up with clear API definitions * Considers batching and streaming if the requirements necessitate that * If appropriate, considers   + authentication/authorization   + idempotence   + latency   + suitability for for caching   + separating low-latency calls that drive UI from high-volume, high-bandwidth API calls for data ingestion.   + if data volumes are large, compression, serialization, and perhaps recovery from failures |  |

**Sample System Design questions**

***Question 1: Online Video Streaming***

**Prompt**

*An influencer that makes and posts videos online wants a way to know their top 10 viewers (by minutes viewed) over the past week. With this functionality, they can send their viewers a thank you note or reward for their loyalty. How would you design this system?*

**NOTES for interviewer:**

* There are multiple questions the candidate should ask to clarify questions and validate assumptions. Based off how the candidate chooses to approach it, there are multiple ways or paths to solve this.
* Distributed systems knowledge would be helpful for non-functional requirements are 1B monthly active users (MAU) and 100k influencers. If you do not have distributed systems knowledge, you can solve this problem without it and get bar raising data points from the candidate aligned to our SDE II calibration guides. Some candidates may be able to solve it really quickly if they have specific experience related to this question but if a candidate does not, that should not be a concerning data point or ding against candidate.
* Nothing about the question assumes Amazon scale, which could put a candidate at a disadvantage to solve as most candidates have not worked at a scale like Amazon. You could assume the system in this example has 100 users and get to a working solution. Can scale up to understand the level of scaling expertise the candidate has.
  + Questions candidates commonly ask to clear ambiguity that an interview should be ready to answer:
* When does a "minute’s view" get recorded? Answer: At the end of every minute or at the end of each view session, but would like to hear tradeoff.
* How accurate does the calculation need to be? Answer: We can start with estimates, but would prefer precise.
* How immediate does the calculation need to be available to the influencer? Answer: Within minutes

**Follow up questions as the interviewer for L4 SDE data points:**

* If candidate is struggling, candidate can walk through at the SDE I level a high level architecture or class design, how would the data flow/API's? This question could be also solved in a coding context by defining class structures so this would be an L4 response.

**Follow up questions as the interviewer for L5 SDE data points:**

* (Operational data point) What parts of your design are critical to add monitors and alarms to ensure on-call is aware of a potential issue?
* (Bar raising L5 data point) How would you adapt your design to support different time ranges (past month or YTD)?
* (Bar raising L5 data point) How would you adapt your design if the requirement were to provide an immediate (within seconds) top 10 calculation to the influencer?

**Follow up questions as the interviewer for L6 SDE data points:**

* How would you adapt your design to address the following use case: A) "Amazon/Google wants a way to know the top 10 viewers by minutes viewed (across all influencers) over the past week. With this functionality, Amazon/Google can send their users a thank you note or reward for their loyalty" B) How would you scale this question from 100 users to 1000 users, etc. C) What are the functional metrics of this system that do not indicate a problem but would be valuable to measure? D) how is failure handles? What if your service is unreachable like you have an outage? How do you recover or design this to recover on it's own?

***Question 2: Online Training Videos***

**Prompt**

*Amazon wants to develop an application that employees can use to browse and watch training and motivational videos.*

**NOTES for interviewer:**

* Candidate will need to ask questions to define the problem statements around trainings, permissions, use cases, etc. It is the same as a "design twitter" only different use case as not everyone knows how Twitter works and we want to stay domain agnostic.
* Question allows candidate to either focus on the external customer focused app + backend or the internal customer website to upload and manage videos + backend. Either way works, however for fungible SDE interviews, gear this questions more towards backend and API design.
* Front end/mobile knowledge will not be needed to do well in this except knowing basic details of knowing a web browser or how a web app works (you have a website, it makes requests to a server which accesses a Database, etc.)
* Some SDE 2 candidates, like robotics or on-chip programmers will have difficulty with this question. Understanding full stack development is needed to be successful with this question.
* This question scales well and can be used to probe for scalability. You can dive super deep on a single aspect of the question like a specific API to scale it down to address L4 SDE level knowledge, or scale up to a highly complicated system to address L6 system design knowledge.
* Requirement to this problem is that the candidate designs an internal way to solve this problem. Using YouTube will not suffice, the interviewer is looking for an in house, secure solution that we have end to end control of.

**Follow up questions as the interviewer for L4 SDE data points:**

* If we wanted to scale the system up a very large amount, what would the cost to operate it scale up by?

**Follow up questions as the interviewer for L5 SDE data points:**

* If the traffic to the system were to increase dramatically 100x or more, where would the system fail first?

**Follow up questions as the interviewer for L6 SDE data points:**

* How would you expand the system to other regions? How would you expand it globally?

***Question 3: Highway Toll Collection System***

**Prompt**

*Design a highway toll collection system. Assume the government provides a system for charging.*

**NOTES for interviewer:**

**Follow up questions as the interviewer for L4 SDE data points:**

* Focus on the API design portion of it. Do not need to go into edge cases, scale it beyond single system, also could assume a service that calculates the rate for you

**Follow up questions as the interviewer for L5 SDE data points:**

* SDE II will want to talk about using APIs for printing/scanning a ticket, generating the charge, and call government for payment. Edge case could be what happens if a ticket is scanned but never retrieved? Prompts SDE II to talk about a stored procedure for "lost tickets".

**Follow up questions as the interviewer for L6 SDE data points:**

* Looking for knowledge on how to scale beyond a toll system, or rate optimization ideas - some will talk about this naturally but interviewer could ask them about it and prompt them, it can be evolved beyond one toll collection system.. Multiple highways..