SRE Design Review Checklist

Some sections may not apply to the system under design - in those cases there is no need to spend time on those sub-checklists.

- ❑ **What and why:** do I understand the need for the change, the design itself, and how it relates to other systems?
- ❑ **Who:** are all affected teams represented in the reviewers, including operations and support?
    - ❑ Privacy and/or security review needed?
- ❑ **Alternatives considered:** is building a new system the right approach? Did the proposer speak to owners of similar systems?
- ❑ **Stickiness:** see sub-checklist
- ❑ **Complexity:** see sub-checklist
- ❑ **Data:** see sub-checklist
- ❑ **Scale and performance:** see sub-checklist
- ❑ **Operability:** see sub-checklist
- ❑ **Robustness:** see sub-checklist

Stickiness sub-checklist

- ❑ What aspects of the design will make major change, or migration and eventual turndown easier or harder?
- ❑ Can users extend the system, with their own code?
- ❑ How tight is the coupling with other systems?
- ❑ What assumptions are baked into the architecture or the data model that might change in the future?

Complexity sub-checklist:

- ❑ Does each component of the system have a clearly defined role and a crisp interface?
- ❑ Is it built using standard building blocks (caches, message queues etc) that engineers at this organisation already understand?
- ❑ Does it use the same kinds of plumbing such as RPC mechanisms, logging, monitoring and so on?

Data sub-checklist:

- ❑ What is the flow of data through the system?
- ❑ What are the data consistency requirements and how does the design support them?
- ❑ What data can be recomputed from other sources and which cannot?
- ❑ Is there a data loss Service Level Objective (SLO)?
- ❑ How long does data need to be retained, and why?
- ❑ Does it need to be encrypted at rest, in transit?
- ❑ Are there multiple replicas of the data?
- ❑ How do we detect and deal with loss or corruption of data?
- ❑ How is data sharded, and how do we deal with growth and resharding?
- ❑ How should data be backed up and restored?
- ❑ What are the access control and authentication strategies?
- ❑ Have relevant regulations such as GDPR and any data residency requirements been addressed?

Scale and performance sub-checklist:

- ❏ What are the bottlenecks in this system that will limit its scale and throughput (not forgetting the impact of writes and locking)?
- ❏ What's the critical path of each type of request, and how do requests fan-out into multiple sub-requests?
- ❏ What is the expected peak load and how does the system support it?
- ❏ What is the required latency SLO and how does the system support it?
- ❏ How will capacity planning and loadtesting be done?
- ❏ What systems are we depending on, what are their performance limits and their documented SLOs?
- ❏ What will it cost to run financially?
- ❏ How will this system deal with a large spike of load?
- ❏ Does the system use caching, and if so, will it be able to serve at increased latency without the cache?
- ❏ Can this system break its backends by making excessive requests?

Operability sub-checklist:

- ❏ How does the design support monitoring and observability?
- ❏ Do all third-party system components provide appropriate observability features?
- ❏ What tools will be available to operators to understand and control the system's behavior during production incidents? How will these tools make clear to the operator what specific actions they will take, to avoid surprises?
- ❏ What routine work is going to be needed for this system? Which team is expected to be responsible for it?
- ❏ Are there manual operations that will be required to recover from common kinds of failure?
- ❏ How to detect and manage abusive users?
- ❏ If the design involves relying on third-parties (such as a cloud provider, hardware or software vendor or even an open-source community), how responsive will vendors be to your feature requests or problems?
- ❏ Are all configurations stored in source control?

Robustness sub-checklist:

- ❏ How is the system designed to deal with failure in the various physical failure domains (device, rack, cluster/AZ, datacenter), plus network partitions or high latency?
- ❏ How could an operator accidentally (or deliberately) break the system?
- ❏ Is there isolation between users?
- ❏ Are hotspots or large shards possible?
- ❏ How can this system degrade gracefully if its dependencies fail?
- ❏ What is the process to restart it from scratch, and how long does that take?
- ❏ Do we depend on anything that might depend on this system?
- ❏ Is the control plane fully separate from the data plane?
- ❏ Can I canary this design effectively?
- ❏ Can this system autonomously drain capacity and how have risks around that been managed, in particular with respect to human operators' ability to understand and control the system?
- ❏ Can this system create self-reinforcing phenomena (i.e. vicious cycles), for example when re-replicating data or retrying?