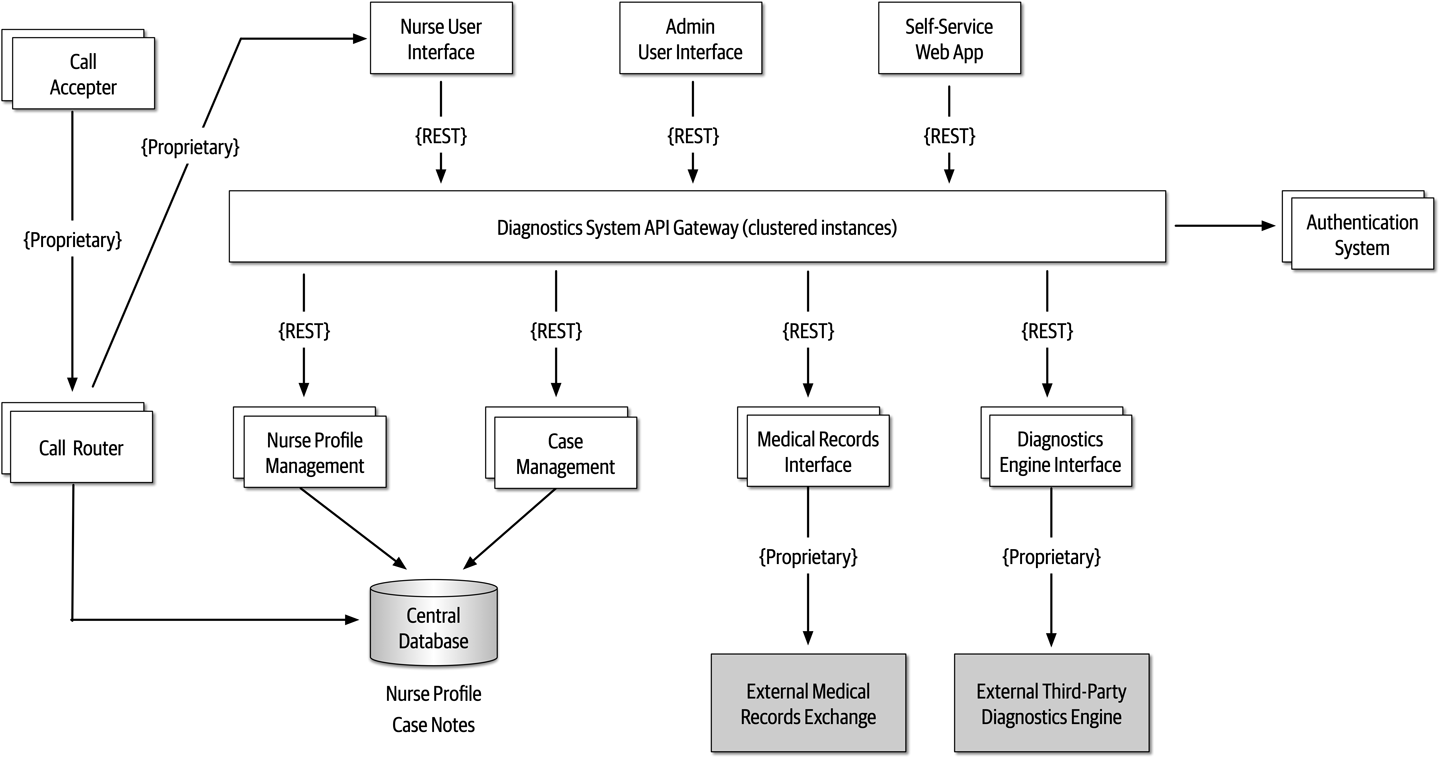
**Scalability case study**

Ref: Fundamentals of software architecture – Mark Richards & Neal Ford Feb 2020 O’Reilly Media

Consider the example of a call center system to support nurses advising patients on various health conditions

* The system will use a third-party diagnostics engine that serves up questions and guides the nurses or patients regarding their medical issues.
* Patients can either call in using the call center to speak to a nurse or choose to use a self-service website that accesses the diagnostic engine directly, bypassing the nurses
* The system must support 250 concurrent nurses nationwide and up to hundreds of thousands of concurrent self-service patients nationwide
* Outbreaks and high volume during cold and flu season need to be addressed in the system.
* The third-party diagnostic engine can handle about 500 requests a second.

Given below is a high level architecture:



Concern:

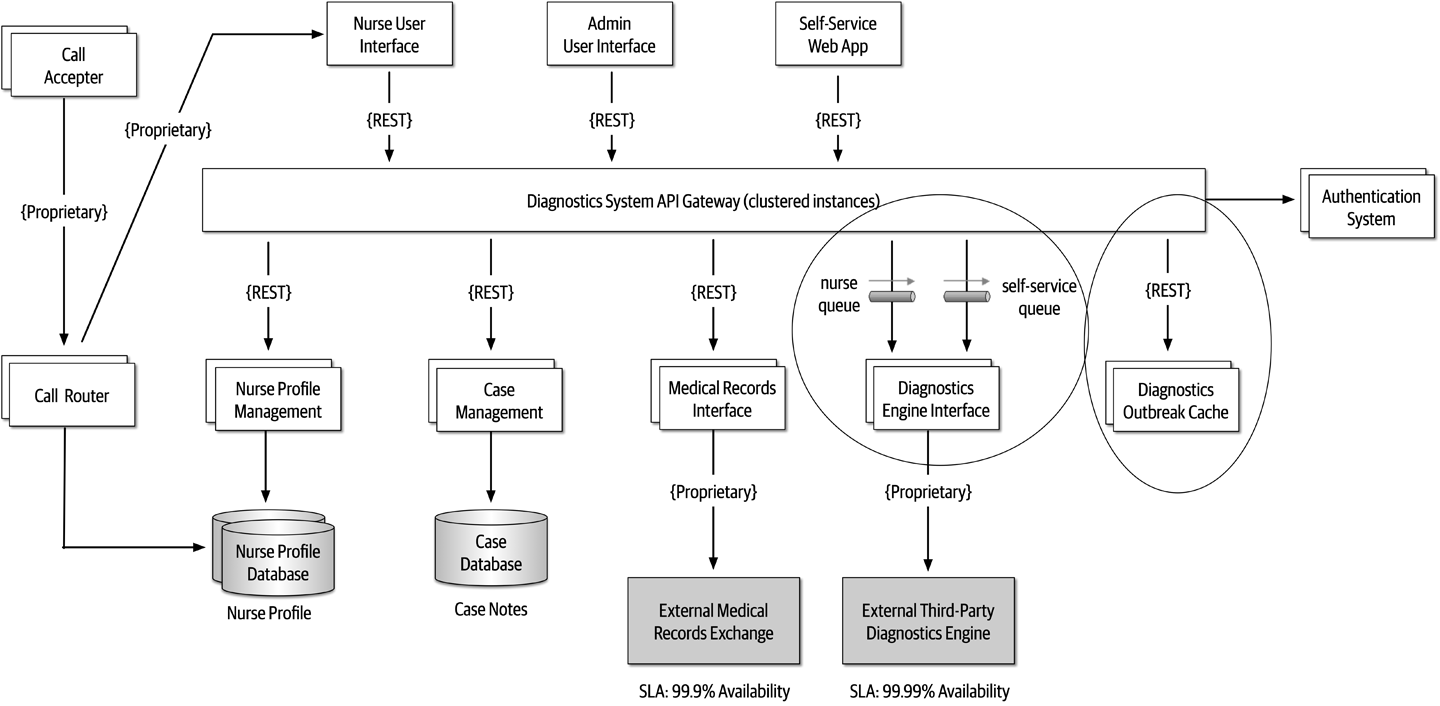
Although there are only 250 nurses, the self-service portion of the system can access the diagnostics engine as well as nurses, significantly increasing the number of requests to the diagnostics interface. Also there was concern about outbreaks and flu season, when anticipated load on the system would significantly increase.

How can the architecture be revised to address these concerns?

Solution:

One way to mitigate this risk is to leverage asynchronous queues (messaging) between the API gateway and the diagnostics engine interface to provide a back-pressure point if calls to the diagnostics engine get backed up. While this is a good practice, it still doesn’t mitigate the risk, because nurses (as well as self-service patients) would be waiting too long for responses from the diagnostics engine, and those requests would likely time out. Leveraging what is known as the [Ambulance Pattern](https://oreil.ly/ZfLU0) would give nurses a higher priority over self-service. Therefore two message channels would be needed. While this technique helps mitigate the risk, it still doesn’t address the wait times. The participants decided that in addition to the queuing technique to provide back-pressure, caching the particular diagnostics questions related to an outbreak would remove outbreak and flu calls from ever having to reach the diagnostics engine interface.

The corresponding architecture changes are illustrated in [Figure 20-12](https://learning.oreilly.com/library/view/fundamentals-of-software/9781492043447/ch20.html#fig-analyzing-architecture-risk-elasticity). Notice that in addition to two queue channels (one for the nurses and one for self-service patients), there is a new service called the *Diagnostics Outbreak Cache Server* that handles all requests related to a particular outbreak or flu-related question. With this architecture in place, the limiting factor was removed (calls to the diagnostics engine), allowing for tens of thousands of concurrent requests. Without a risk storming effort, this risk might not have been identified until an outbreak or flu season happened.



**Figure 20-12. Architecture modifications to address elasticity risk**