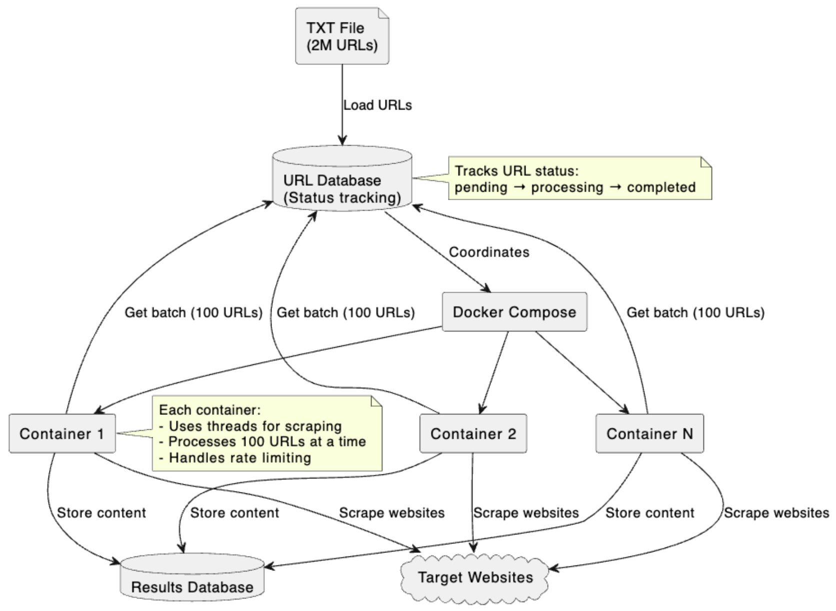
To address the challenge of scraping 2 million websites daily, my initial approach was to design a multi-container Docker system. In this setup, I envisioned two separate database containers: one for storing the list of URLs to be scraped and another for saving the resulting web content. Alongside these databases, I planned to deploy several worker containers. Each container would be responsible for fetching a batch of URLs from the database, scraping the respective webpages, and then sending the extracted content to the results database for storage. The diagram below provides a conceptual visualization of this architecture.



While this distributed, containerized system would certainly work, upon further reflection I realized that it might be overkill for the task at hand. Managing a system of multiple containers introduces a fair amount of complexity, particularly when considering fault tolerance. I would need to configure container orchestration tools to monitor and restart failed services, and inside each container, I'd have to write logic to handle scraping failures, such as timeouts, unexpected responses, and network errors. This would add a considerable operational burden to what is essentially a high-volume but relatively simple scraping task.

In contrast, a monolithic application offers a much more straightforward and efficient alternative. Rather than maintaining two separate databases, I would consolidate everything into a single database that stores the scraped content. Instead of spawning multiple containers, I would rely on threads (or processes, depending on the hardware) within a single application to achieve parallelism. The logic would involve using a loop to retrieve batches of URLs, each of which would then be handed off to a thread. Once a thread completes its scraping task, it would insert the results into the central database. This monolithic approach is fast, resource-efficient, and significantly easier to implement and debug. While I generally favor microservices for large-scale, modular systems, in this case, the simplicity and cohesion of a monolith is more appropriate and better aligned with the task’s requirements.