MONITORING AND PROFILING

WICHAT_ES2A

Software Architecture - University of Oviedo

Natalia Blanco Agudín
Marcos Llanos Vega
David Covián Gómez
Darío Cristóbal González
Hugo Fernández Rodríguez
Hugo Prendes Menéndez

Grafana and Prometheus

Grafana and Prometheus are widely used tools for monitoring and visualizing the performance of systems and applications. Prometheus is a time-series based monitoring platform that collects, stores, and allows querying of metrics using its powerful PromQL query language. Grafana, on the other hand, is a visualization tool that integrates seamlessly with Prometheus to graphically display collected metrics, making it easier to interpret data in real time through interactive dashboards. In this project, we will use Prometheus to gather metrics related to the application's behaviour—such as resource usage, response times, or error rates—and Grafana to visualize this data clearly and effectively, enabling us to detect anomalies, analyse performance, and make informed decisions to improve the system.

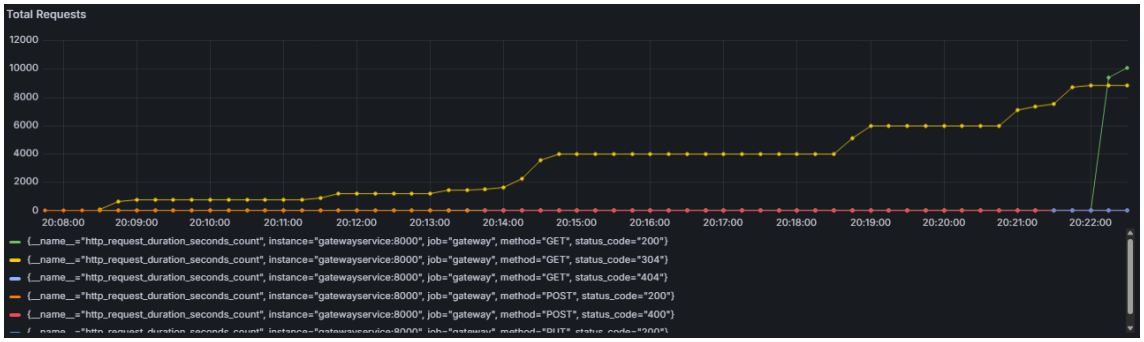
Metrics

The first thing we must ensure is that Prometheus is collecting the data needed to generate our metrics. In Grafana, we can check its status over time using the “UP” metric, which is set to 1 if the service is working properly.

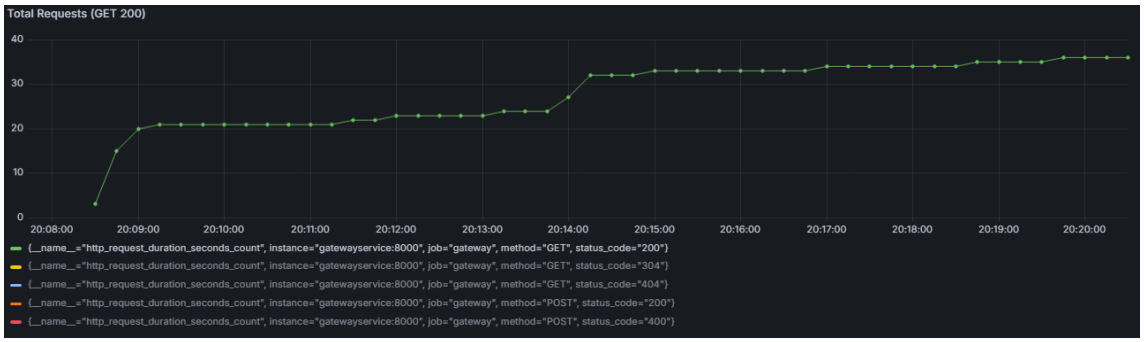
Now that we are sure about Prometheus working correctly, and before going into our project’s details, we can even check how much time Prometheus needed to scrape data from our app. It should be noted that these results are extracted from a local and controlled environment, which explains the fast response times as the system is not subjected to “real” traffic and load. In a production environment, however, it would make more sense to look at this metric to ensure stable operation of Prometheus.

We now move on to analyse the results of the monitoring of our application. For this purpose, we have recreated a normal situation of use of the application (create users, add friends, play games, consult histories, send messages, use groups, ...) and finished with an overload simulation, launching thousands of GET requests to the service to see how it responded. This context is reflected in the following graph, which shows the total

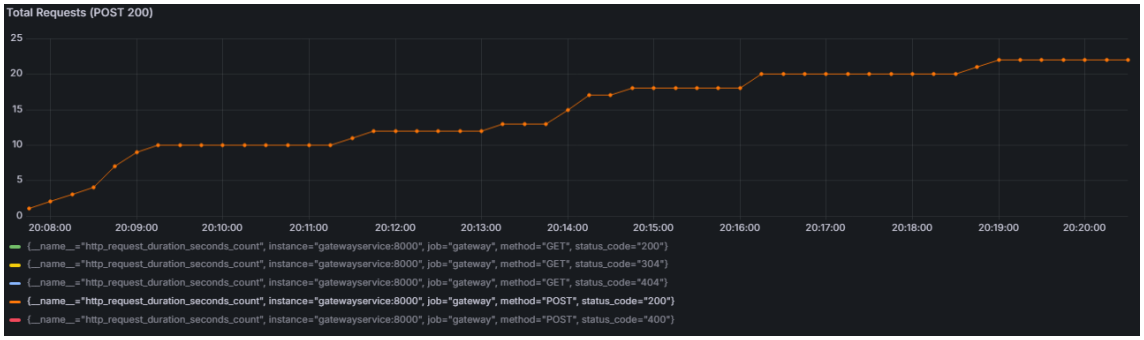
number of HTTP requests of each type (GET/POST), grouped also depending on their status (if successful or not).



Going into more detail, we can analyse the number of successful GET requests that have been made (we did not include the overload in order to observe actual performance on an appropriate scale, but its magnitude, and the correct response the system offered to it, can be seen in the graph above by following the green line).

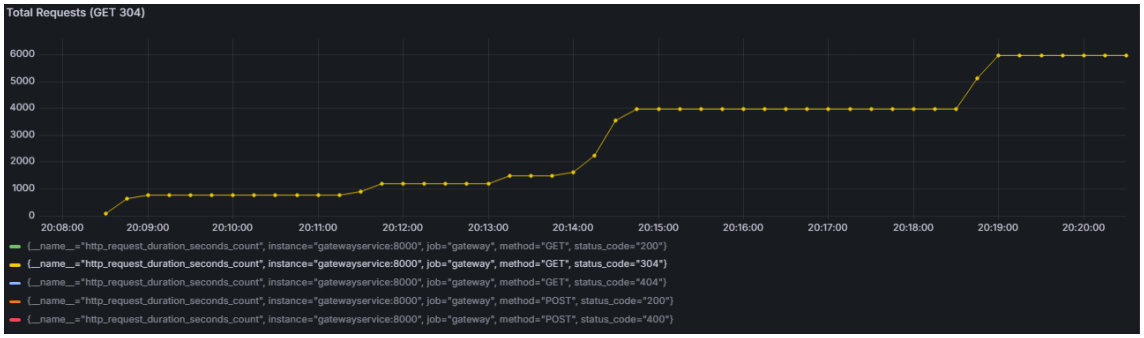


Also, successful POSTs.

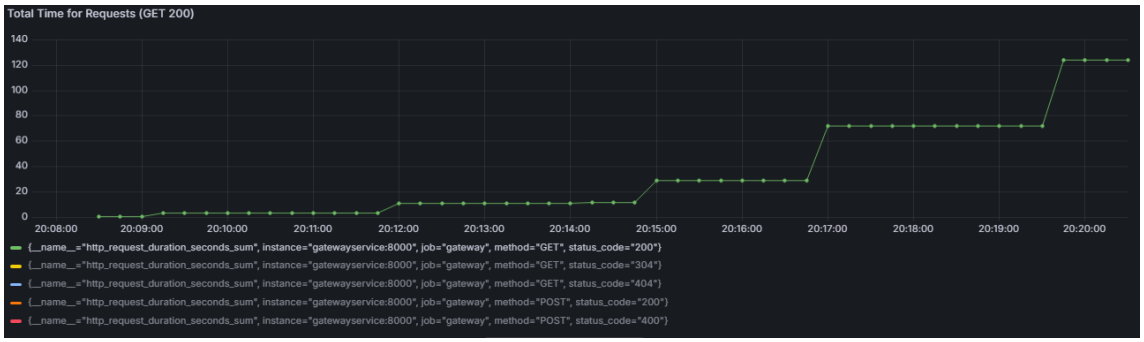


Unsuccessful POST requests (only one in our tests, intentionally made by sending a message without content to the global chat).

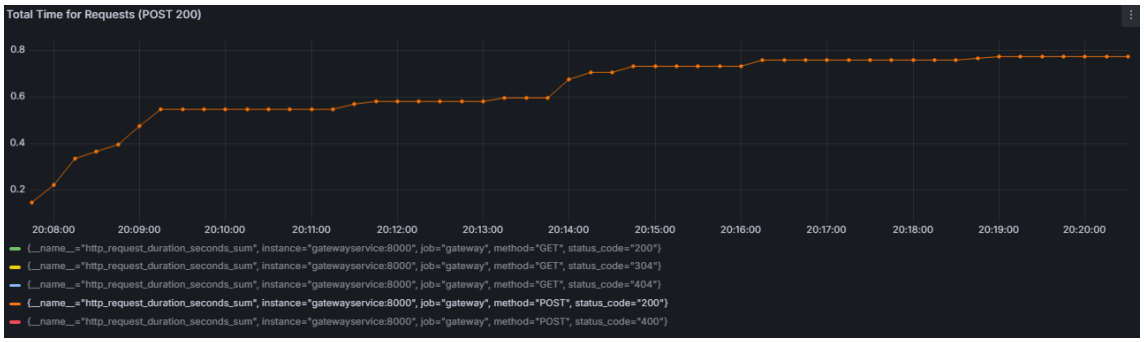
And, finally, we can also consult the number of times that something cached has been consulted by capturing GET requests with code 304.



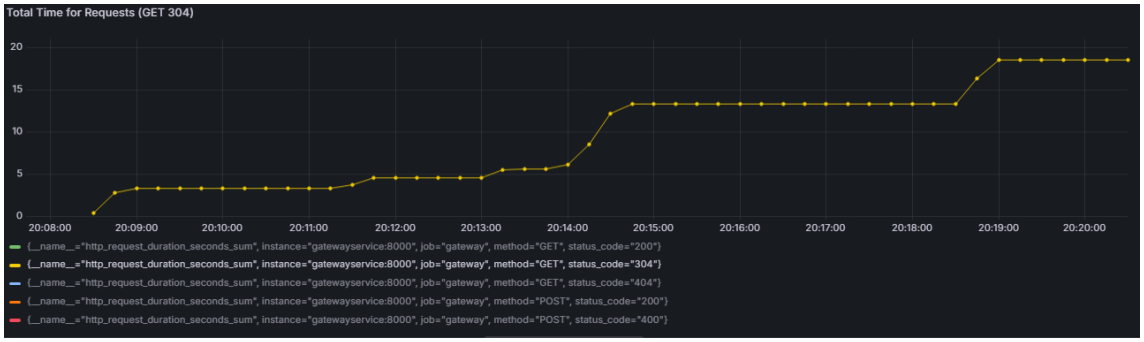
Grafana and Prometheus also provide us the total time required for the system to respond to the different kind of requests. For example, we can check the time required to respond to successful GET requests (overload excluded). In this regard, it's worth highlighting how the wait time required to start a game increases the graphics values, affecting performance. We detected this issue and are currently awaiting resolution.



Also, the time taken to react to the successful POSTs.



And finally, the one needed to retrieve data from the cache.



Once the results have been analysed, we can extrapolate the conclusion that the application has a stable performance (GET and POST requests of all types are mostly successful and offer relatively low response times, although there is room for improvement, particularly in terms of game loading times) and supports even large request overloads. It is also important to highlight the large number of requests that are made on cached resources, something that could weigh down the performance of the system and that requires an investigation by the development team to determine its causes.