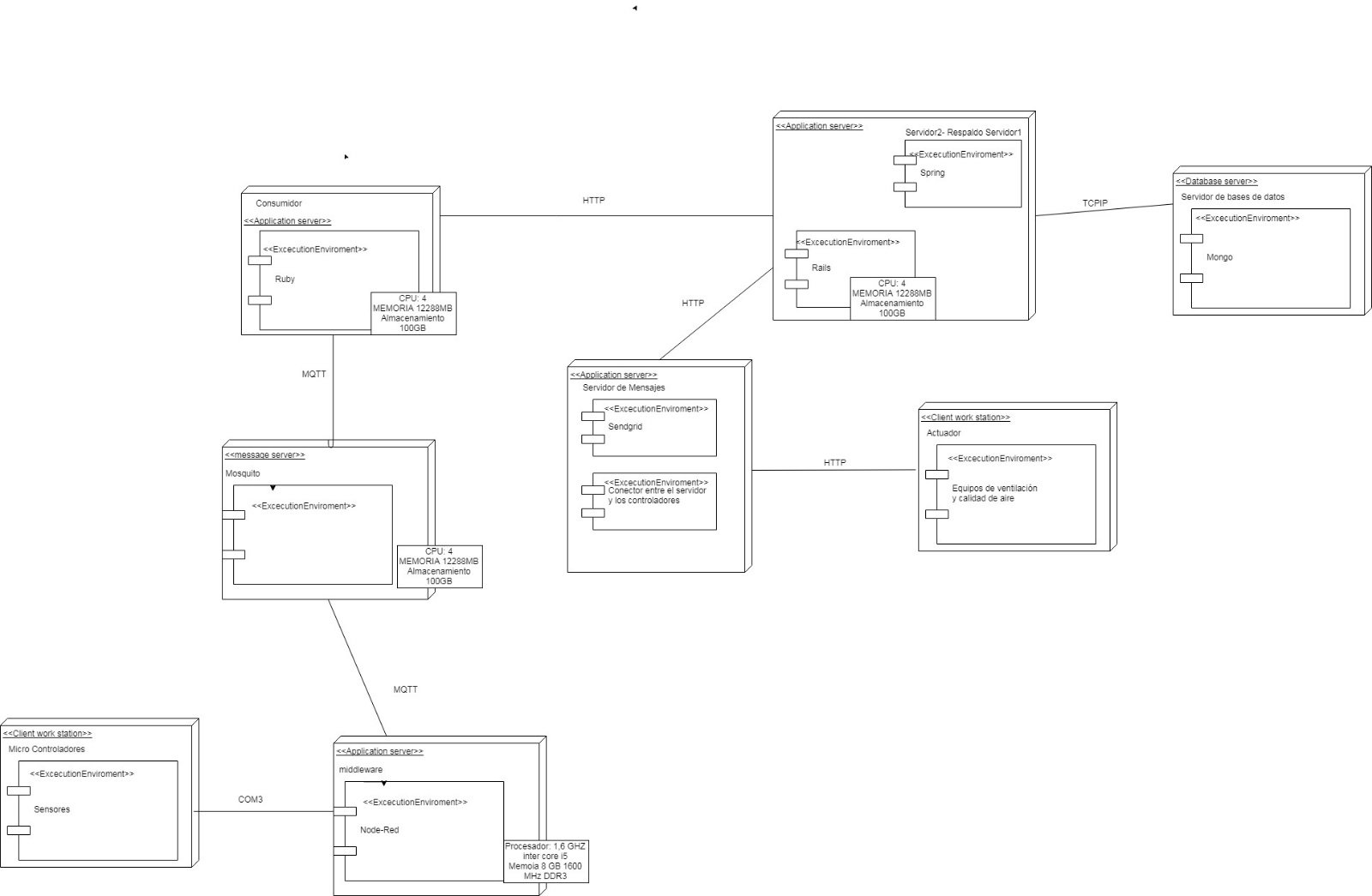
**Diagrama de componentes**



**Analisis Grafico**

|  |
| --- |
| GET |
| Etiqueta | # Muestras | Media | Mín | Máx | Desv. Estándar | % Error | Rendimiento | Kb/sec | Sent KB/sec | Media de Bytes |
| registro medicion | 10 | 13 | 10 | 16 | 1.95192213 | 0 | 0.185126904 | 0.06689156 | 0.03525366 | 370 |
| registro medicion | 100 | 13 | 7 | 51 | 7.002142529 | 0 | 1.682708488 | 0.6080099 | 0.32043765 | 370 |
| registro medicion | 500 | 13 | 7 | 62 | 5.557056415 | 0 | 8.335834084 | 3.0119713 | 1.58739028 | 370 |
| registro medicion | 1000 | 17 | 7 | 232 | 15.02015443 | 0 | 16.61129568 | 6.00212832 | 3.16328385 | 370 |
| registro medicion | 5000 | 19 | 6 | 293 | 32.17808433 | 0 | 81.63798452 | 29.4980999 | 15.5462959 | 370 |
| registro medicion | 9500 | 1305 | 80 | 2530 |  | 1,17 |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Post | | | | | | | | | | |
| Etiqueta | # Muestras | Media | Mín | Máx | Desv. Estándar | % Error | Rendimiento | Kb/sec | Sent KB/sec | Media de Bytes |
| registro medicion | 10 | 14 | 10 | 18 | 2.952841296 | 0 | 0.185126904 | 1.06689516 | 0.03533366 | 350 |
| registro medicion | 100 | 20 | 10 | 44 | 7.002456729 | 0 | 1.847084876 | 0.3544608 | 0.33333765 | 350 |
| registro medicion | 500 | 11 | 5 | 70 | 7.557065642 | 0 | 8.885834084 | 5.0119735 | 2.58736789 | 350 |
| registro medicion | 1000 | 17 | 7 | 232 | 18.00433544 | 0 | 14.67329957 | 9.67903223 | 6.35658385 | 350 |
| registro medicion | 5000 | 26 | 15 | 303 | 36.19758433 | 0 | 81.64322144 | 14.4793999 | 12.789033 | 350 |
| registro medicion | 9500 | 1000 | 90 | 6570 |  | 0,57 |  |  |  |  |

**Análisis de diseño**

Para esta entrega se decidió hacer ciertos cambios en la arquitectura utilizad. Para comenzar se reemplazó el lenguaje de programación a Ruby on rails, gracias a este cambio se eliminaron los componentes de Kafka, zookeper ,  el bridge  y el consumidor los cuales  que se había planteado anteriormente en java y python. La base de datos utilizada para la anterior entrega e fue mongoDB la cual se mantuvo.

Estas decisiones se tomó basado en que Ruby on rails tiene beneficios como que es un lenguaje de programación es más amigable, fácil de usa, flexible, necesita menor tiempo de implementación y parametrización re  reutilizar componentes existentes .

Complementario a esto este lenguaje de programación a pesar de que se podría considerar simple ofrece todas las funciones y beneficios que cualquier otro lenguaje como Java o python que eran los que se estaba utilizando anteriormente y además combina características de lenguajes dinámicos con soluciones de lenguajes estáticos, junto con esto las aplicaciones realizadas en Ruby on rails son modificables con mayor facilidad  por esto es que Ruby se considera un lenguaje apuntado a la productividad  y se podría decir que favorece la escalabilidad.

La base de datos de mongo se mantuvo gracias a que es una base de datos relacional favorece la escalabilidad, es más flexible, se pueden realizar cambios en el proyecto sin tener que realizar cambios drásticos en la base de datos y las consultas de grandes cantidades de datos son más eficientes, cualidad que se necesita en este proyecto porque se maneja un volumen de datos bastante extenso dado a que se realizan mediciones cada 60 o 120 segundos.

Como se puede apreciar en las gráficas el proyecto cumple con las necesidades de eficiencia solicitada tanto para el método get como para el post, esto se pudo lograr gracias a los cambios de diseño descritos anteriormente.