Experimento 1 Entrega final

Grupo: SoftSecurity

Integrantes:

Zulma Lorena Castañeda Hidalgo

Tatiana Vanessa Huertas Bolaños

Andrés Felipe Pinzón Adame

Juan Camilo Useche Rodriguez

Arquitectura y diseño de software
2018-1

Universidad de los Andes

1. **Base de datos: Escenario de prueba**

Para cumplir con el requerimiento de persistencia de los elementos del problema (conjuntos, inmuebles, hubs y cerraduras) decidimos hacer uso de una base de datos no relacional documental como lo es MongoDB, optamos por este estilo de base de datos pues buscamos que en el proyecto exista máxima velocidad a la hora de acceder a la información. En nuestra base de datos tenemos un documento por cada una de estas entidades con los datos básicos (identificación, nombre, dirección) y exactamente un hub. Cada cerradura tiene un hub asociado a través del inmueble.

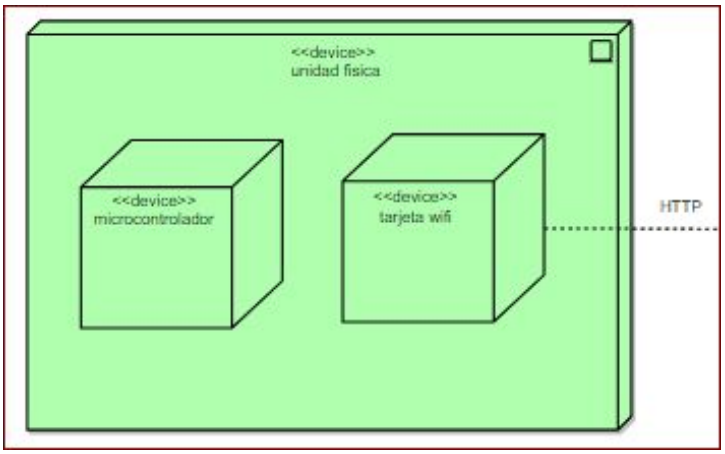
Para realizar las pruebas del API REST, el cual desarrollamos haciendo uso de Python y el framework Flask, poblamos las tablas con 10 Conjuntos, 100 inmuebles y hubs con sus 1000 cerraduras por cada uno de ellos. Registramos también información de las alarmas en total 10000.

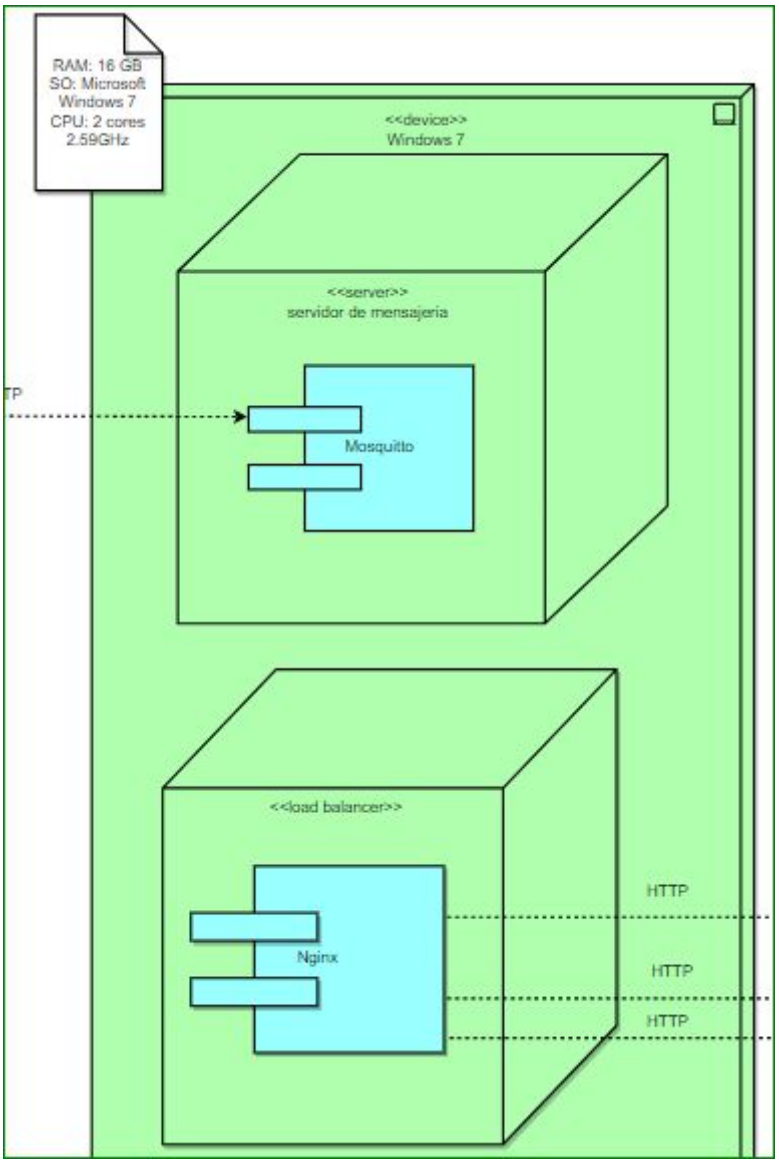
Estas inserciones se realizaron gracias a Flask sin errores o problemas.

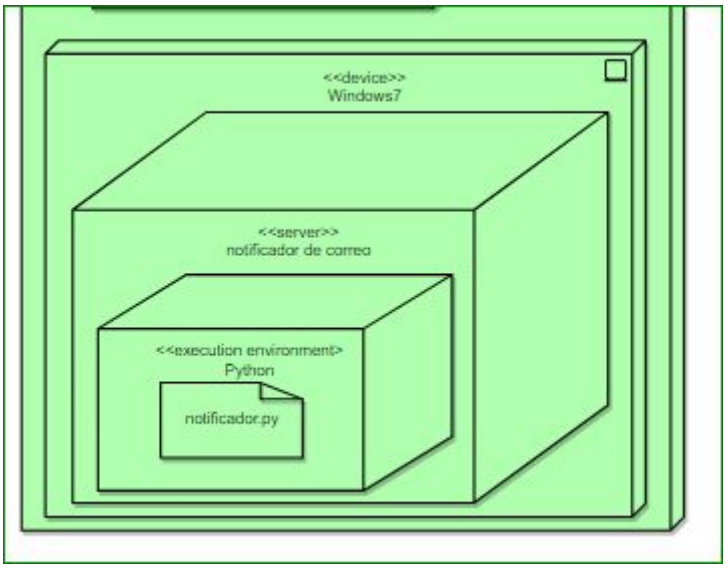
2. **Diagrama de despliegue**

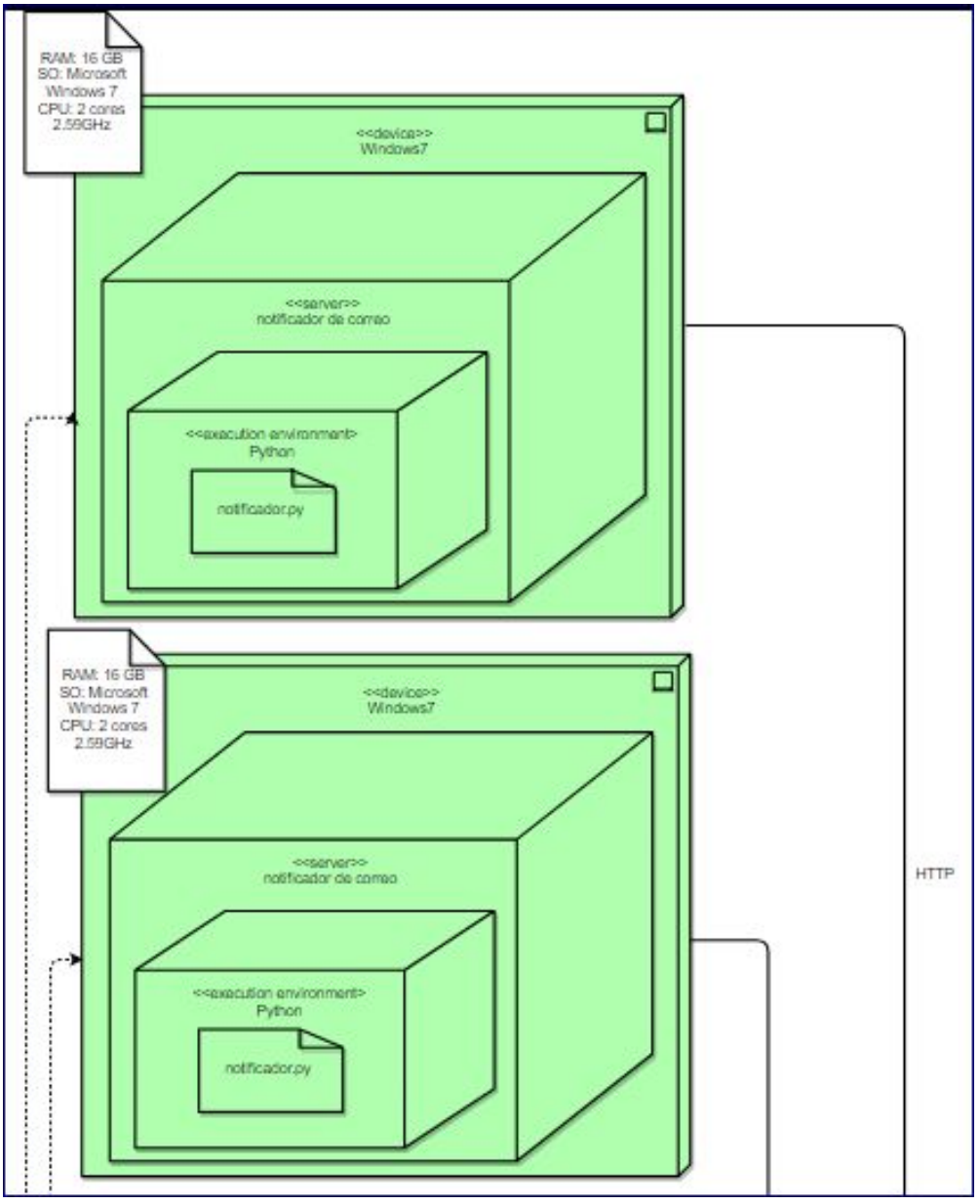
Se realizó el siguiente diagrama de despliegue indicando para cada nodo su capacidad. Entre los nodos principales encontramos: la unidad física y 4 nodos que corresponden a las máquinas virtuales del grupo. En el nodo de la unidad física encontramos las entidades de microcontrolador y la tarjeta wifi. Es la tarjeta wifi quien se conecta con la máquina en la que se encuentra el servidor de mensajería a través de un protocolo HTTP en el puerto 8083. En ese mismo nodo se encuentra también el balanceador de carga que distribuye la carga entre ese mismo nodo y los nodos que corresponden a las otras máquinas virtuales. Por lo tanto, se encuentra en esta y en las otras máquinas un notificador de mensajes, simulando un servidor de correo: recibiendo la petición, imprimiendola en consola y generando una respuesta satisfactoria. Por otro lado, se seleccionó una de las máquinas para allí destinar el manejo de la persistencia. Es en esta máquina donde se encuentran los servicios REST que permitirán invocar la persistencia.

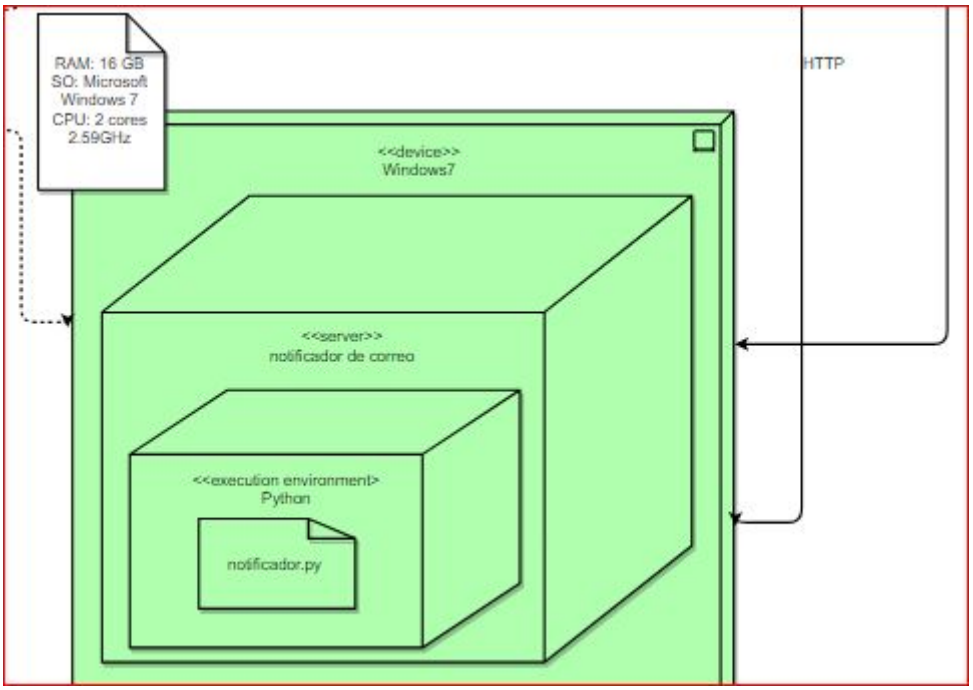
Para la persistencia, nuevamente, se hizo uso de la base de datos no relacional documental mongoDB. Debe de haber una conexión entre los notificadores de correo de las otras máquinas y aquella que maneja la persistencias de modo que cada una pueda persistir las alarmas que les fueron distribuidas por el balanceador de carga. Por último, cabe mencionar que la comunicación entre los nodos servidor se da mediante un protocolo HTTP.

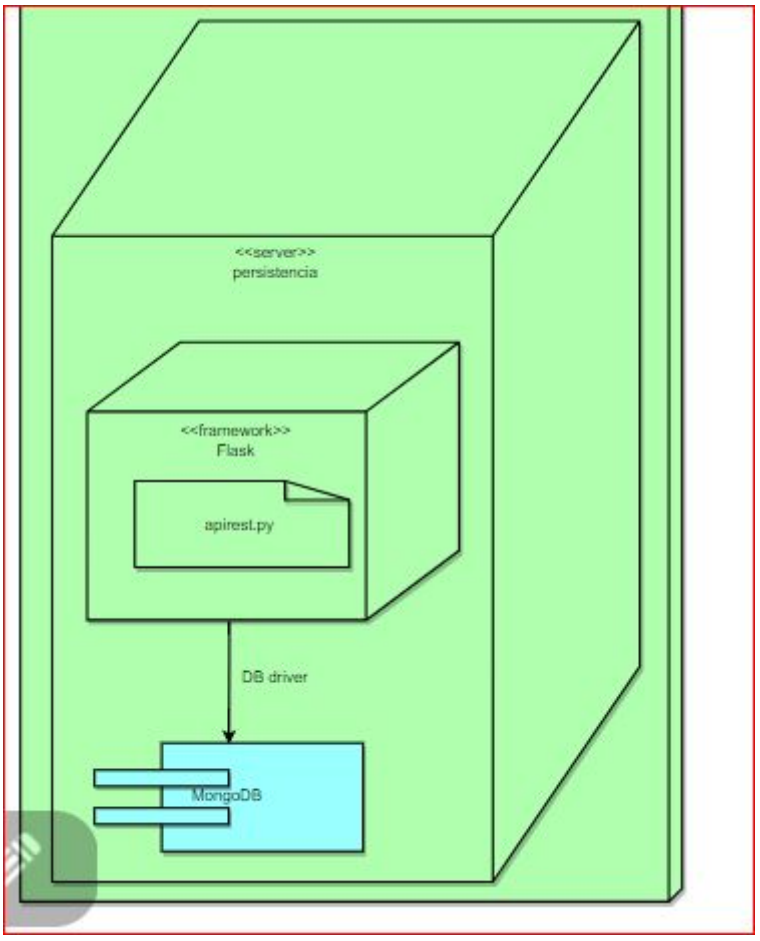

<<device>>
unidad fisica
<<device>>
microcontrolador
<<device>>
tarjeta wifi
HTTP


RAM: 16 GB
SO: Microsoft Windows 7
CPU: 2 cores 2.59GHz
<<device>>
Windows 7
<<server>>
servidor de mensajeria
TP
Mosquitto
<<load balancer>>
Nginx
HTTP
HTTP
HTTP

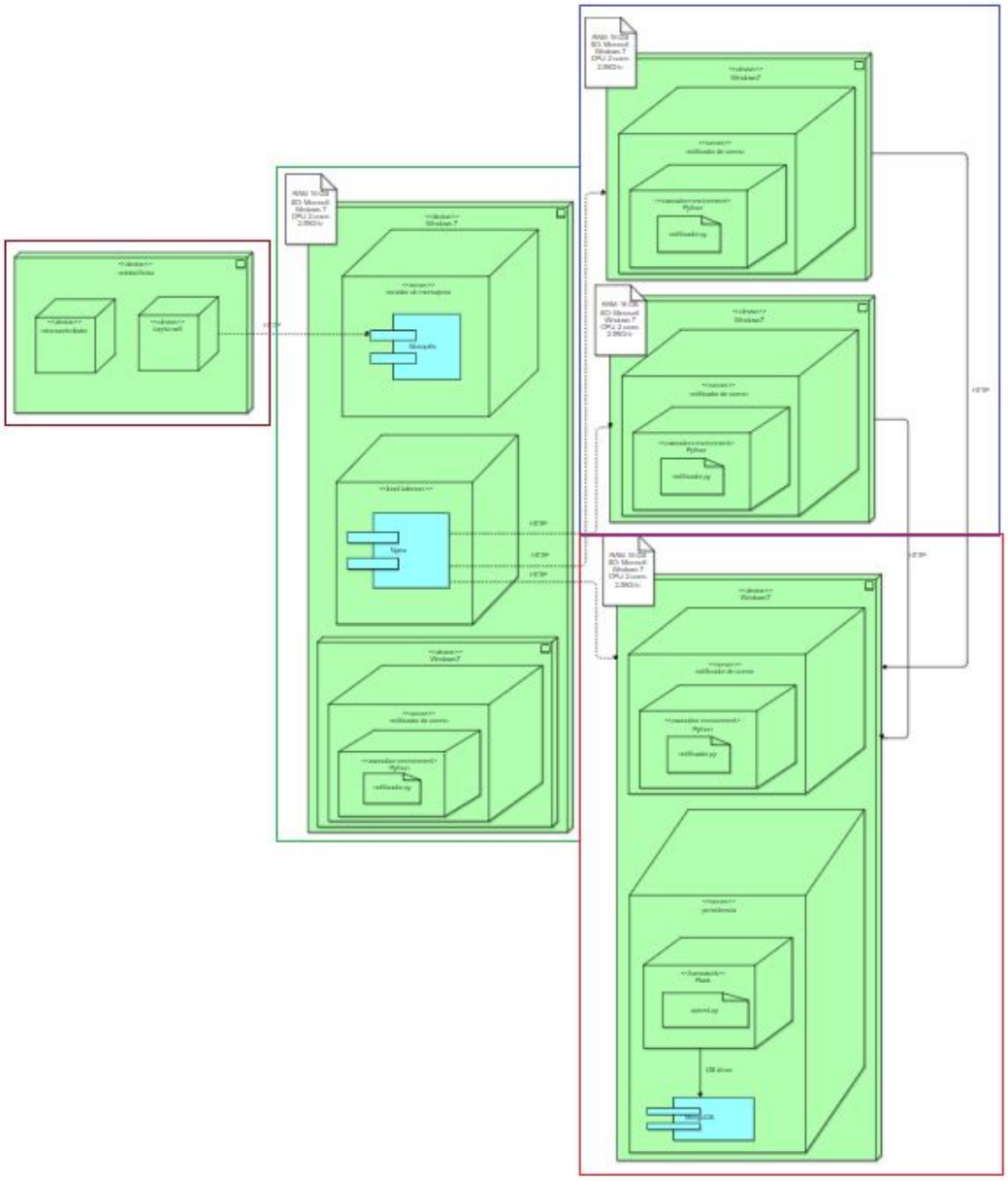

<<device>>
Windows7
<<server>>
notificador de correo
<<execution environment>>
Python
notificador.py


RAM: 16 GB
SO: Microsoft Windows 7
CPU: 2 cores 2.59GHz
<<device>>
Windows7
<<server>>
notificador de correo
<<execution environment>>
Python
notificador.py
RAM: 16 GB
SO: Microsoft Windows 7
CPU: 2 cores 2.59GHz
<<device>>
Windows7
<<server>>
notificador de correo
<<execution environment>>
Python
notificador.py
HTTP





Vista general:



3. Comparación de resultados

Para las pruebas de carga decidimos utilizar JMeter, el escenario de calidad solicitaba que la aplicación pudiera soportar en el escenario más crítico de alarmas provenientes de las 1500 casas ubicadas en los 200 conjuntos que plantea tener YALE para 2020, es decir 300.000 operaciones que se envían con una ventana de tiempo de 1 minuto con un tiempo de respuesta inferior a 1 segundo. Para verificar el escenario realizamos pruebas de carga mediante Apache Jmeter.

Resultados de las pruebas de carga:

Realizamos pruebas sobre los servicios REST creados para la prueba con 200 inmuebles tuvimos un tiempo promedio de 5 ms con 0% de error.

Para la prueba de las alarmas se probó en 4 maquinas con 75000 peticiones cada una (en total 300000), retornando un error del 57.53% y una media de 3 segundos de respuesta. El máximo número de peticiones alcanzado dentro de los parámetros fue de alrededor de 20000 por cada máquina (80000 peticiones en total).

4. Reflexión sobre el diseño actual de la arquitectura

En el desarrollo del experimento se eligió python como tecnología en programación. Lo encontramos bastante eficiente en tiempo, líneas de código y riesgo de cometer errores. Por otro lado, creemos que hacer uso de una arquitectura play podría haber traído mejores resultados al proyecto, gracias a su característica no bloqueante, dado que sería posible recibir y procesar múltiples mensajes simultáneamente aumentado la eficiencia del proyecto. Sin embargo, su implementación resultó complicada dados nuestros conocimientos actuales.

Flask es un framework de python simple y fácil de implementar que nos ayudó a ahorrar mucho tiempo a la hora de desarrollar, sin embargo al realizar las pruebas de carga pudimos darnos cuenta de que su desempeño no es el mejor (100.000 inserciones con un porcentaje de error del 37%) debemos implementar una táctica que nos permita mejorar esta métrica, bien sea migrando el API REST a un framework más robusto como Django o implementar varias instancias del programa en diferentes máquinas con un balanceador de carga.