Réalisation d’une application de type RAG

**      **

Certification : Développeur d’application en Intelligence Artificiel

Marseille - Nice

Bloc de compétences 3 - E5

Réaliser une application intégrant un service d’intelligence artificielle

LOMBARDI Joachim

Table des matières

[A9. Compétence : Maintien en Condition Opérationnelle (MCO) d'une application d'IA 3](#_Toc193109685)

[C20. Surveillance et Monitorage de l'Application 3](#_Toc193109686)

[1) Métriques de la performance du système 3](#_Toc193109687)

[2) Métriques de la performance du RAG (métriques personnalisées) 5](#_Toc193109688)

[3) Alertes 7](#_Toc193109689)

[4) Monitorage avec uptime-kuma 9](#_Toc193109690)

[C21. Résolution d'Incidents Techniques 10](#_Toc193109691)

[Conclusion 12](#_Toc193109692)

# A9. Compétence : Maintien en Condition Opérationnelle (MCO) d'une application d'IA

## C20. Surveillance et Monitorage de l'Application

Afin de réaliser le monitorage de mon application, je vais utiliser les métriques suivantes :

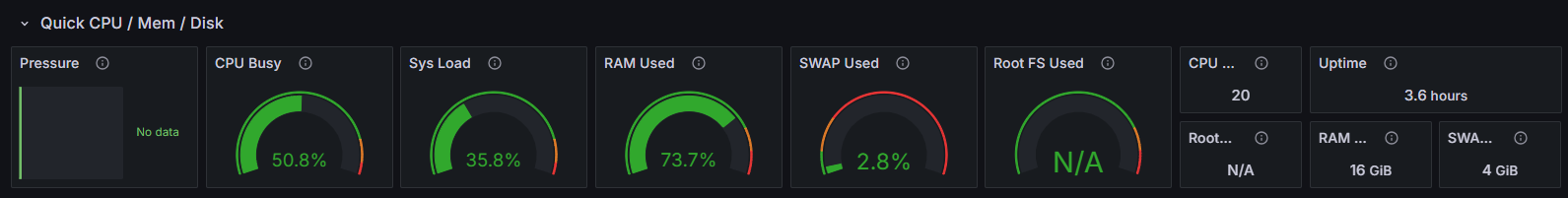
* rag\_pipeline\_latency : Latence du pipeline RAG (tout le processus).
* search\_latency : Latence de la recherche documentaire.
* llm\_latency : Latence de génération LLM.
* rag\_requests\_total : Nombre total de requêtes RAG.
* rag\_errors\_total : Nombre total d'erreurs RAG.
* rag\_success\_ratio : Ratio de succès des requêtes (Gauge car c'est une valeur calculée).
* rag\_error\_type\_total : Nombre d'erreurs par type (retrieval, llm ou unknown).

Ainsi que des métriques plus générales :

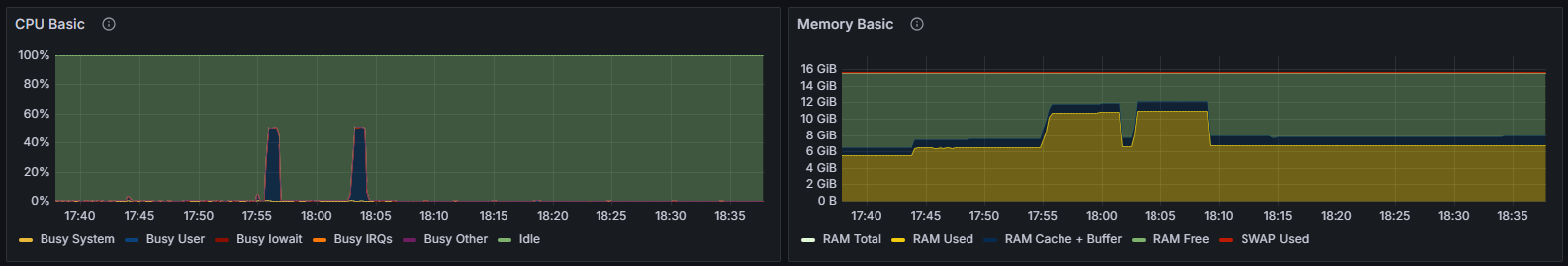
* node\_cpu\_seconds\_total : Temps total passé par le CPU dans différents états (idle, user, system…)
* node\_memory\_MemAvailable\_bytes : Mémoire disponible (sans swap)
* node\_memory\_SwapFree\_bytes : Espace de swap disponible
* node\_disk\_io\_time\_seconds\_total: Temps total passé en I/O (lecture/écriture).
* node\_network\_receive\_bytes\_total : Nombre total d’octets reçus par l’interface réseau.
* node\_network\_transmit\_bytes\_total: Nombre total d’octets envoyés.

Comme indiqué dans le rapport E3, le monitorage est réalisé par prometheus et grafana

### Métriques de la performance du système



*Figure 1. Tableau des principales mesures de la performance du système*



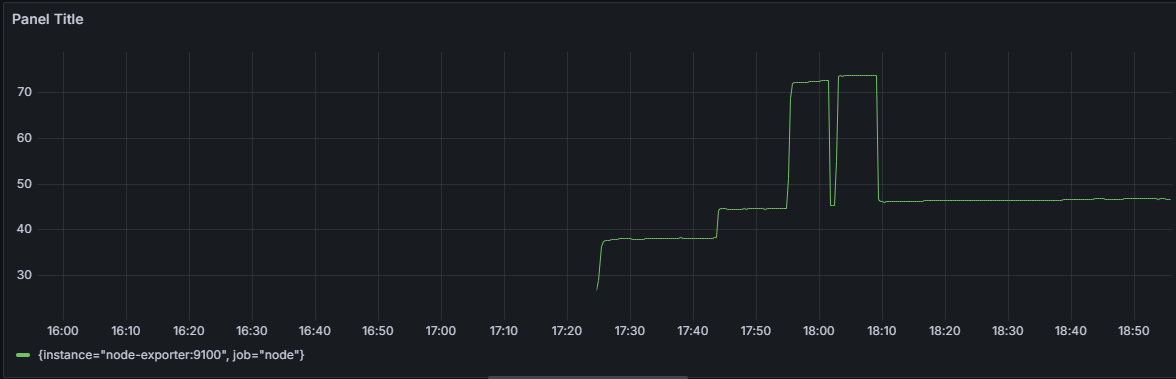
*Figure 2. Graphe d'utilisation du CPU et de la mémoire*

* Busy : Temps global du processeur occupé.
* Low : Faible activité ou peu de charges sur le CPU.
* System : Temps utilisé pour les processus système (noyau).
* IRQ : Temps que le processeur passe à gérer les interruptions matérielles.
* Idle : Temps où le processeur est inactif et ne traite aucune tâche.
* SWAP : zone de stockage sur le disque dur (ou SSD) utilisée lorsque la mémoire vive (RAM) d'un système est pleine.



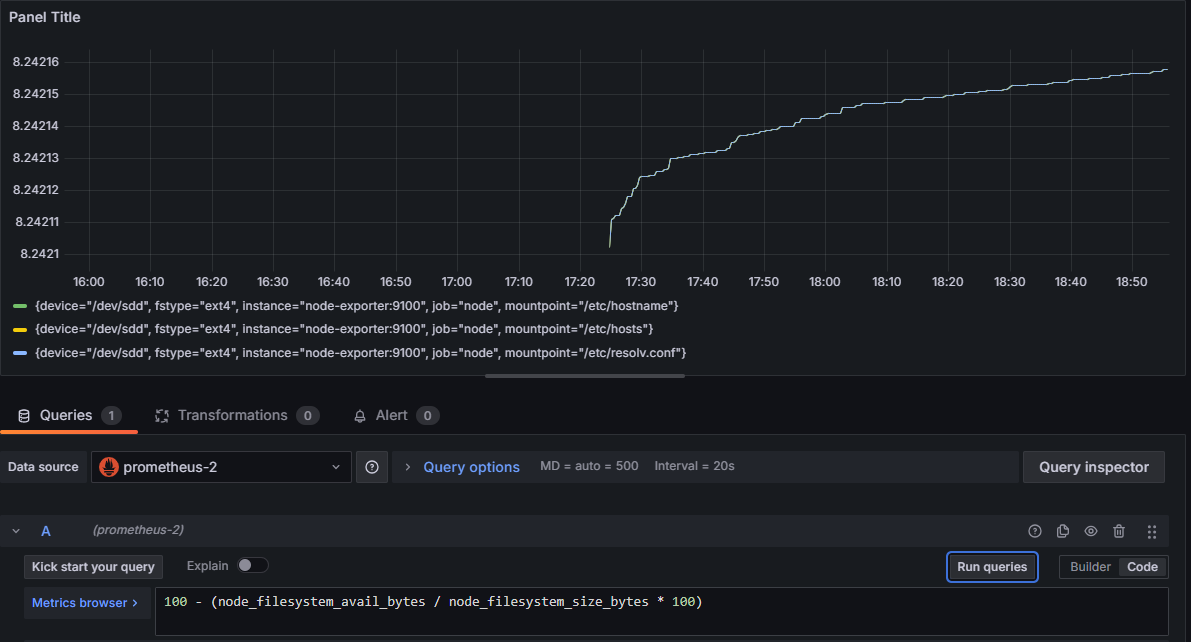
*Figure 3. Charge CPU*

100 - (avg(rate(node\_cpu\_seconds\_total{mode="idle"}[5m])) \* 100)

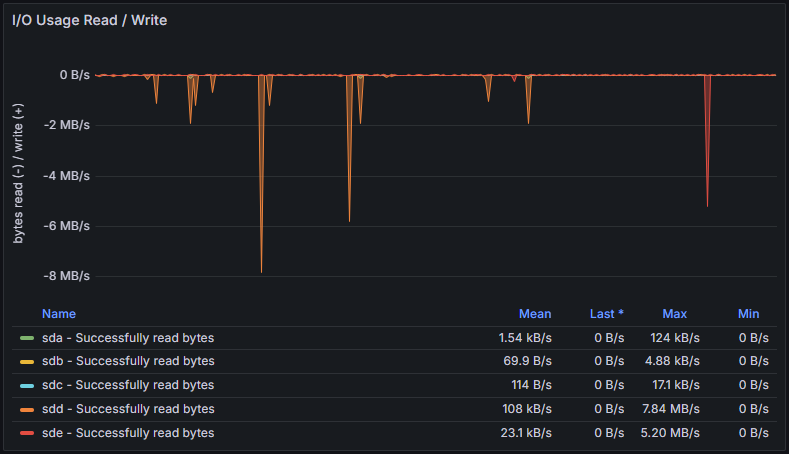


*Figure 4. Mémoire utilisée*

(node\_memory\_MemTotal\_bytes - node\_memory\_MemAvailable\_bytes) / node\_memory\_MemTotal\_bytes \* 100

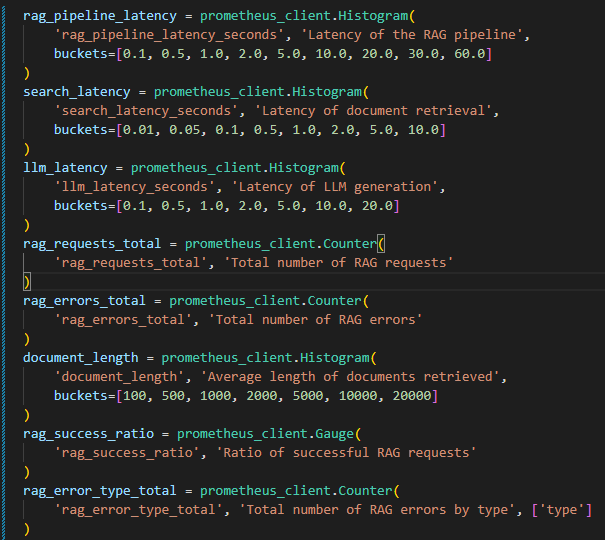


*Figure 5. Utilisation du disque*



*Figure 6. Utilisation du disque Lecture/Ecriture.*

### Métriques de la performance du RAG (métriques personnalisées)



*Figure 7. Script de mes métriques personnalisées*

Un bucket (ou "seau" en français) est une catégorie ou un intervalle dans lequel les données sont regroupées lors de l'analyse et de la visualisation.

Dans le contexte de Grafana et d'Elasticsearch :

Un bucket peut représenter une plage de temps (ex. : toutes les données par tranches de 5 minutes).

Il peut aussi correspondre à un comptage (ex. : nombre de documents correspondant à une requête).

Les buckets permettent donc d'agréger et de structurer les données pour mieux les analyser et les afficher sous forme de graphiques ou de tableaux.

Dans ce cas, une seule requête a été effectuée. Par conséquent, afficher les courbes avec des buckets n’apporte pas d’information pertinente.

**

*Figure 8. Durée de la recherche*

**

*Figure 9. Durée de la génération*



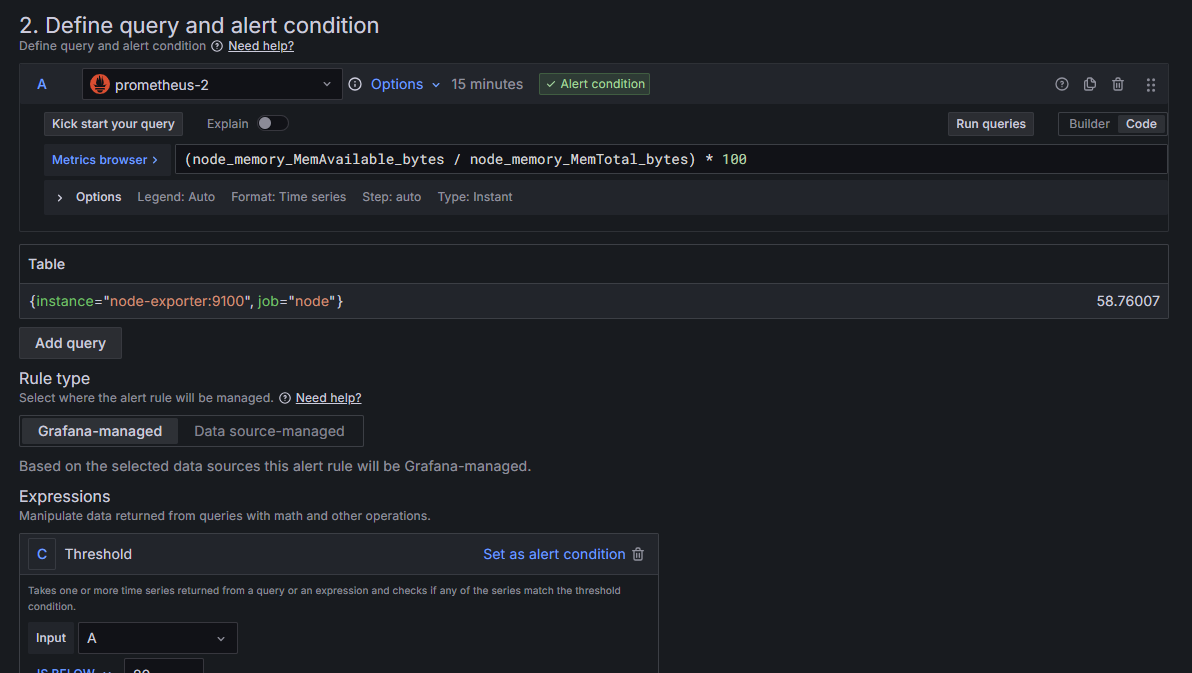
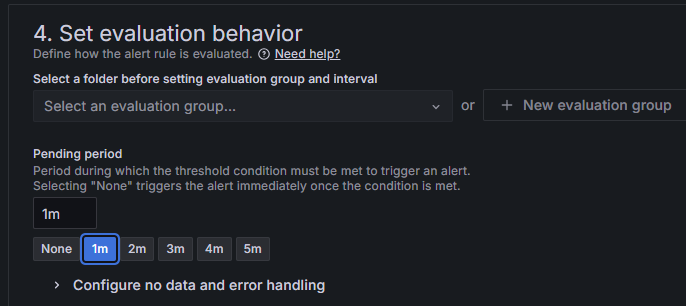
*Figure 10. Pourcentage de rag ayant marché*

|  |
| --- |
|  |
| *Figure 11. Durée totale du RAG* | | | |
| *Figure 12. Nombre total de requête* | *Figure 13. Nombre total d'erreur du RAG* | *Figure 14. Pourcentage de rag ayant marché* |

### Alertes

On peut mettre des alertes.

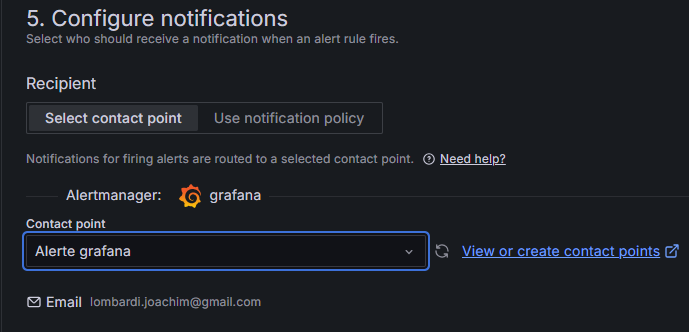
Ici lorsque la RAM disponible est inférieure à 20 % durant les 15 dernières minutes.

*Figure 15. Formulaire de création d'alerte pour la RAM Figure 16. Formulaire pour définir la période durant laquelle le seuil doit être franchie*  *pour déclencher une alerte*

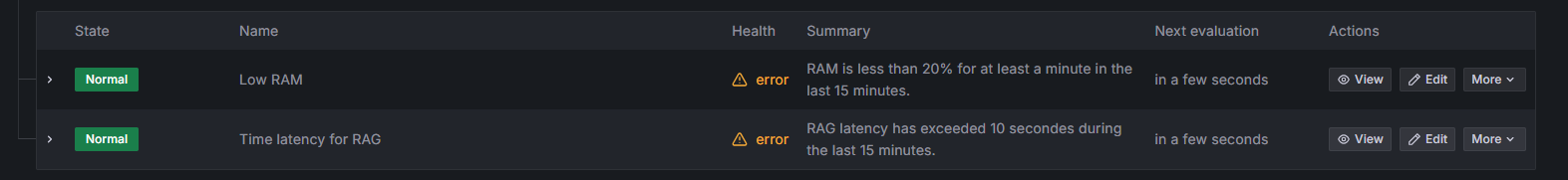
Ici le temps durant lequel le seuil doit être atteint pour déclencher une alerte.

Ici, on configure l’email :



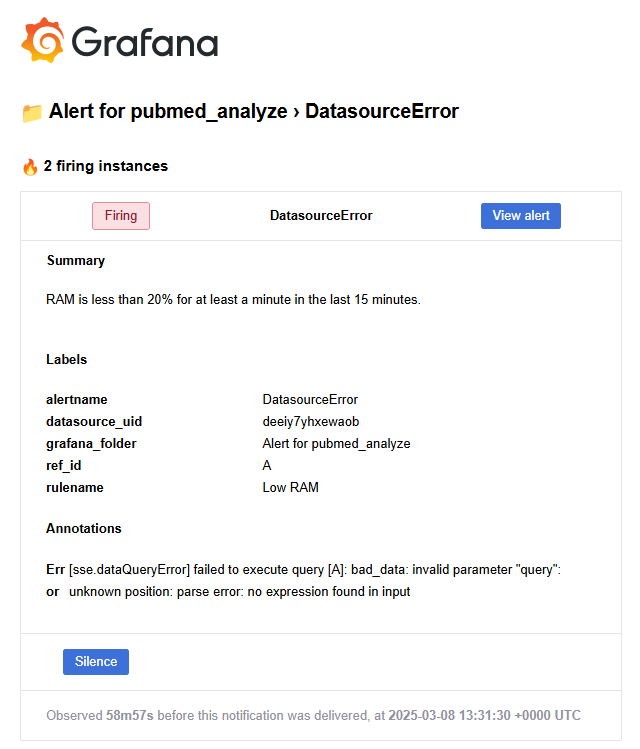
*Figure 17. Selection de l'email d'alerte*

J’ai mis une deuxième alerte si le temps d’exécution du RAG est supérieur à 10 secondes durant les 15 dernières minutes.

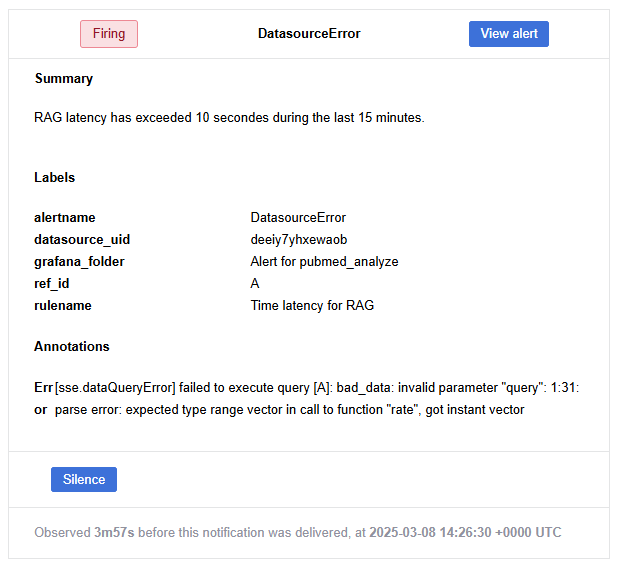


*Figure 18. Liste des alertes avec le statut*

Ici mes deux alertes m’indiquent que les seuils que j’ai établis ont été franchis.



*Figure 19. Alerte mail Grafana envoyée suite à la consommation de RAM de plus de 80%*

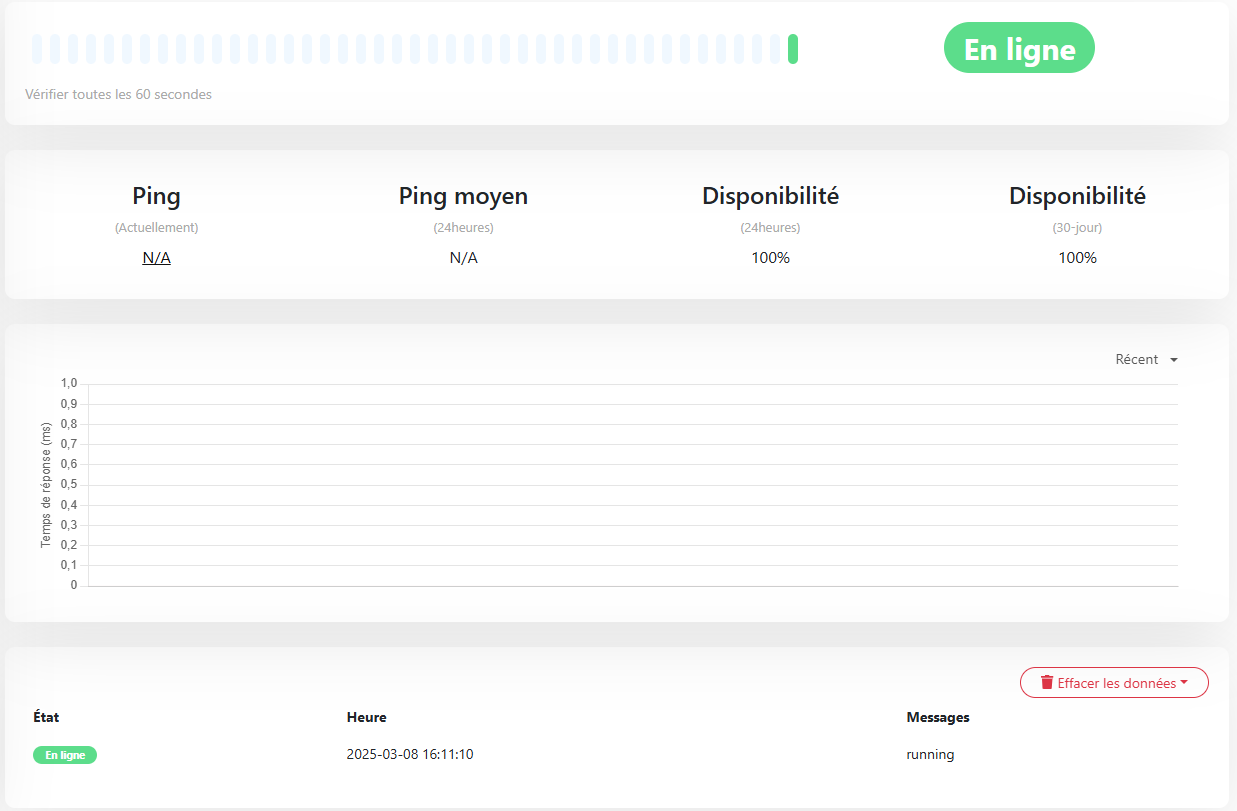


*Figure 20. Alerte mail Grafana envoyée suite à la latence du RAG de plus de dix secondes.*

### Monitorage avec uptime-kuma

La surveillance que j’ai mis en place pour uptime-kuma est pour détecter si les conteneurs sont actifs par l’envoie de mail.

Uptime-kuma permet également de visualiser l’état des conteneurs via un tableau de bord.



*Figure 21. Tableau de bord d'Uptime-kuma*

Ici on voit que le conteneur est en ligne.

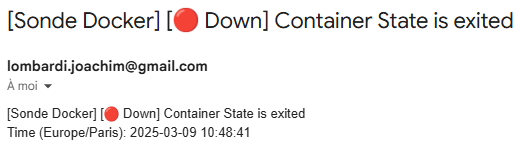
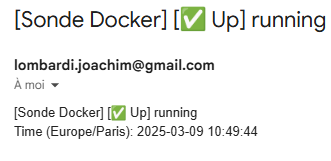
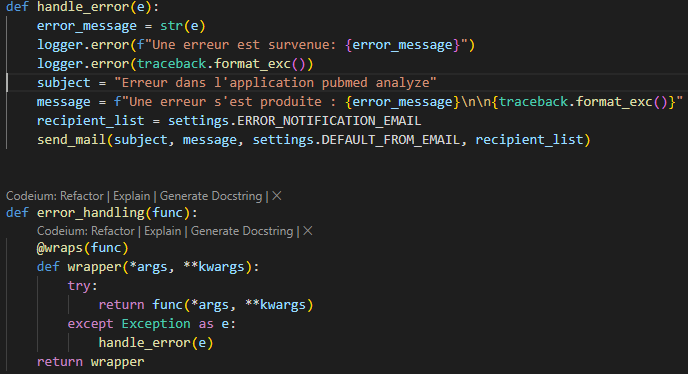
 

Figure 22. Email d'alerte pour le docker hors ligne. Figure 23. Email pour indiquer que le conteneur est en ligne

## C21. Résolution d'Incidents Techniques

Afin de gérer mes erreurs, j’ai créé une fonction dans Django dans le fichier utils.py qui « attrape » les erreurs de mon application et me les envois par email.

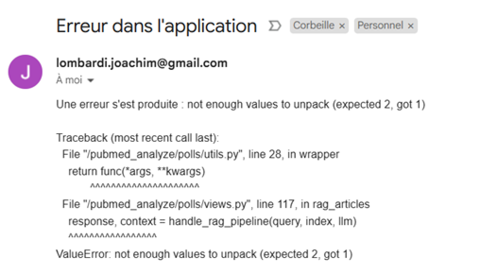


*Figure 24. Script pour gérer les erreurs.*

1. @wraps : Préserve la fonction d’origine.
2. Wrapper est exécutée à la place de func, elle gère les erreurs en plus d’exécuter la fonction func.
3. Logger enregistre les erreurs dans un fichier errors.log.
4. Traceback récupère la « trace » de l’erreur.

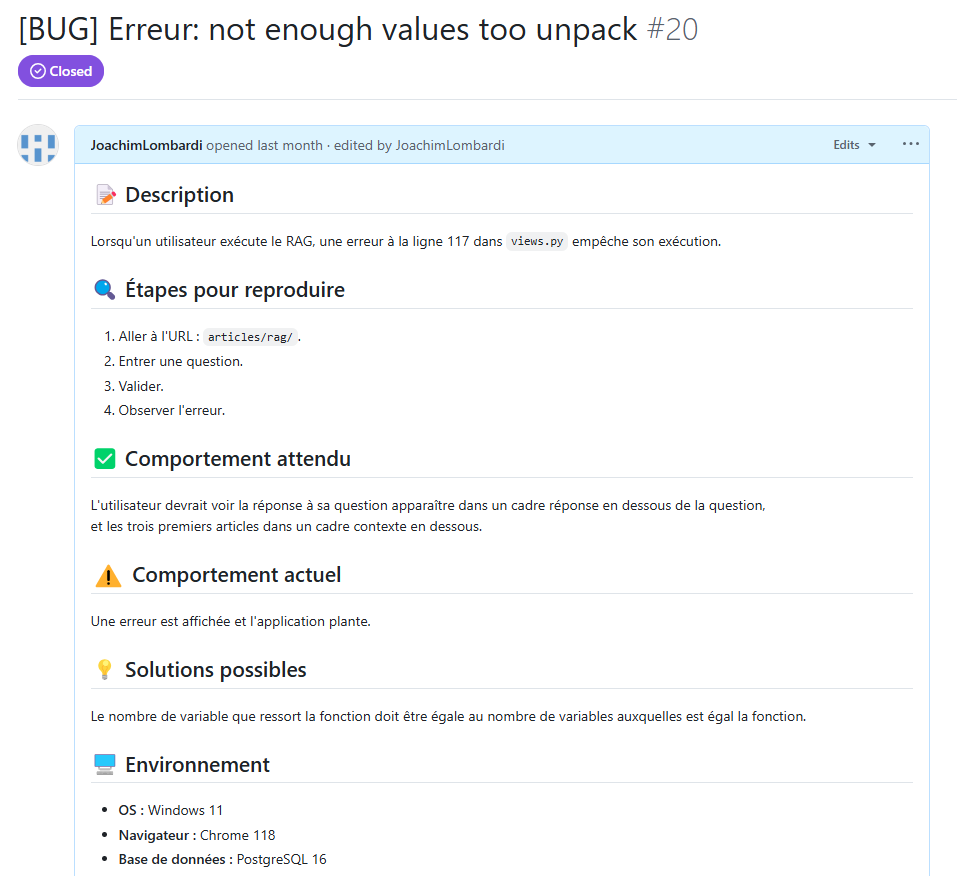
Exemple d’une erreur.

Lorsque mon programme a été exécuté, j’ai reçu l’email suivant :



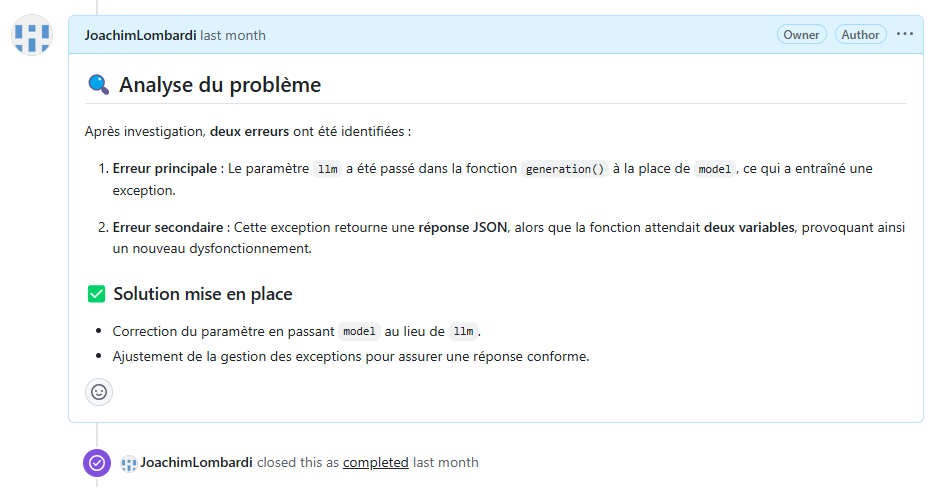
*Figure 25. Email d'une erreur.*

J’ai donc ouvert un ticket sur GitHub.



*Figure 26. Ticket sur GitHub*

L’erreur indique que deux valeurs sont attendues mais qu’il n’en a eu qu’une. Ainsi, la fonction *handle\_rag\_pipeline* renvoie une valeur en sortie mais deux variables sont proposées. Après analyse, il y avait deux erreurs, la première entraînant l’exception à savoir le mot « LLM » a été passé dans les paramètres de la fonction *generation* à la place de « model ». Ce qui a entraîné une exception mais cette exception renvoie une réponse JSON et non deux variables. Après correction de l’erreur, j’ai répondu au ticket en expliquant l’origine de l’erreur et sa solution et je l’ai fermé.



*Figure 27. Résolution du ticket sur GitHub.*

# Conclusion

Nous avons défini les métriques de surveillance de note application, aussi bien des métriques systèmes (consommation du CPU, RAM…) que des métriques propres au projet (latence du RAG, recherche, LLM…). La surveillance de l’état de l’application (hors ligne ou non). Des alertes pour indiquer lorsque la consommation de RAM ou la latence du RAG est trop élevé. Enfin un cas de résolution d’erreur.