Architecture Microservices de DeepSeek

DeepSeek AI

September 26, 2025

1 Objectif du document

Analyser et modéliser l'architecture de DeepSeek et proposition d'une amélioration de l'architecture.

2 Première hypothèse

Cette analyse est proposée par DeepSeek en lui demandant son architecture et la liaison entre ses composants.

2.1 Architecture Générale

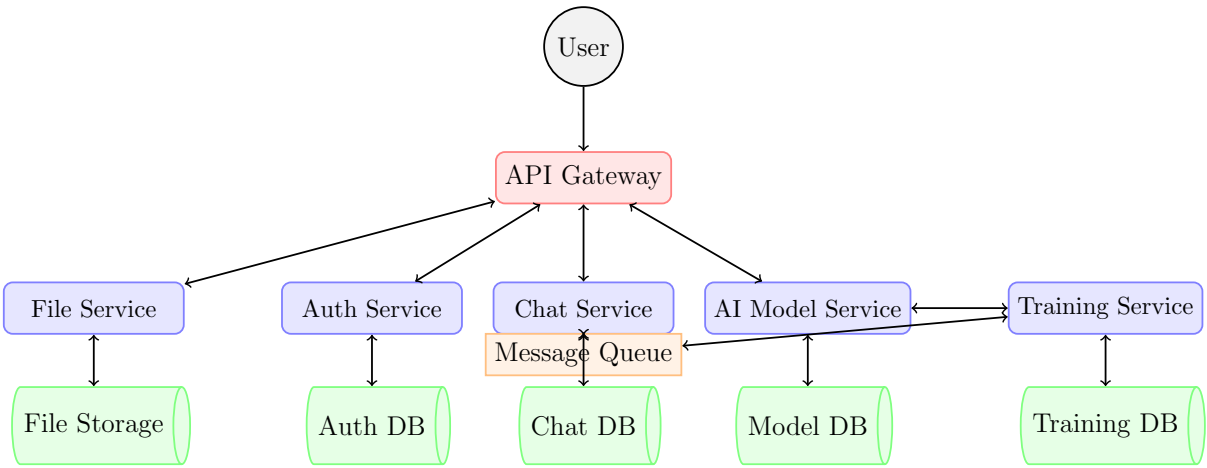

Figure 1: Architecture Microservices de DeepSeek

2.2 Flux de Communication

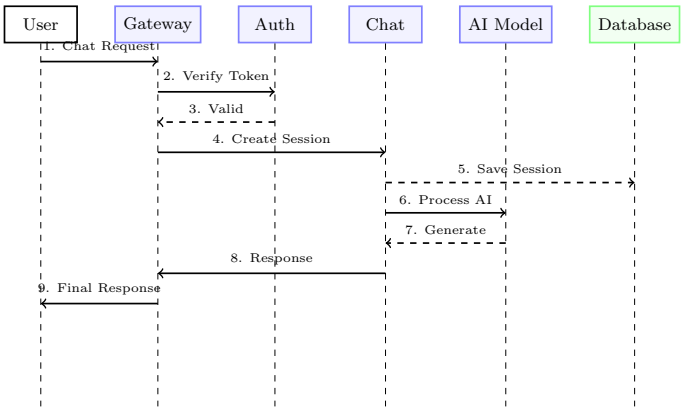

Figure 2: Diagramme de Séquence - Processus de Chat

2.3 Composants des Services

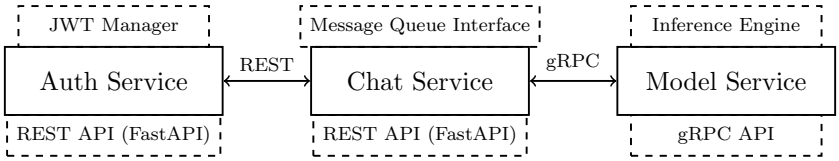

Figure 3: Diagramme des Composants des Services Principaux

2.4 Description des Services

Services Principaux

- **API Gateway**: Routeur principal, load balancing, rate limiting, SSL termination
- **Auth Service**: Gestion des utilisateurs, JWT tokens, OAuth2, permissions
- **Chat Service**: Gestion des conversations, sessions, historiques, streaming
- **AI Model Service**: Inférence des modèles LLM, génération de texte, embeddings
- **Training Service**: Fine-tuning, entraînement des modèles, optimisation
- **File Service**: Upload/download, stockage cloud, gestion des documents

Services Support

- **Message Queue**: RabbitMQ/Kafka pour communication asynchrone
- **Monitoring**: Prometheus, Grafana pour métriques et alertes
- **Logging**: ELK Stack pour centralisation des logs
- **Cache**: Redis pour cache distribué et sessions
- **Service Discovery**: Consul pour découverte automatique des services

2.5 Spécifications Techniques

Stack Technologique

Composant	Technologies
Conteneurisation	Docker, Kubernetes, Helm
Communication	gRPC, REST API, WebSocket, RabbitMQ
Bases de Données	PostgreSQL, Redis, Weaviate (vector DB)
Monitoring	Prometheus, Grafana, Jaeger
CI/CD	GitHub Actions, ArgoCD
Sécurité	OAuth2, JWT, Vault, TLS/mTLS

Table 1: Stack technologique utilisé

Patterns d'Architecture

- **API Gateway Pattern**: Point d'entrée unique pour toutes les requêtes
- **Database per Service**: Isolation des données par service
- **Event-Driven Architecture**: Communication asynchrone via messages
- **Circuit Breaker**: Résilience aux pannes des services dépendants
- **CQRS**: Séparation des lectures et écritures pour les données critiques
- **Saga Pattern**: Gestion des transactions distribuées

2.6 Architecture Détaillée du Moteur d'Inférence

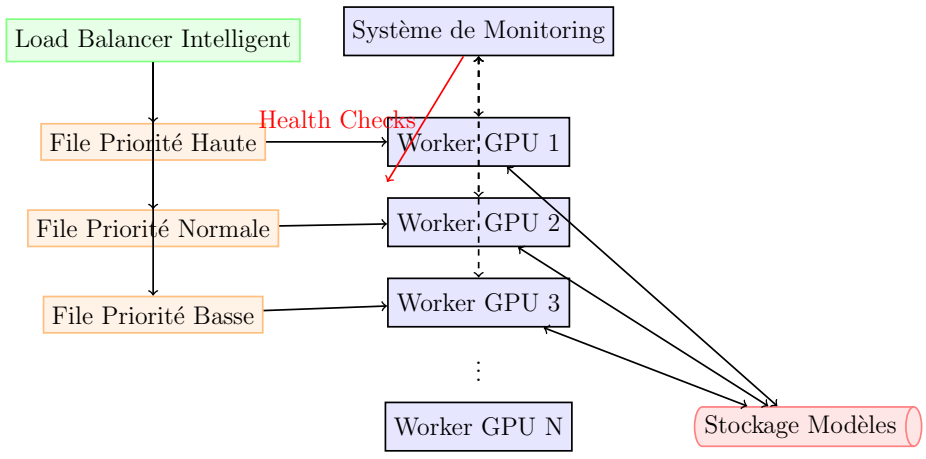

Figure 4: Architecture du Moteur d'Inférence avec Mécanismes Anti-Blocage

2.7 Mécanismes de Prévention des Blocages

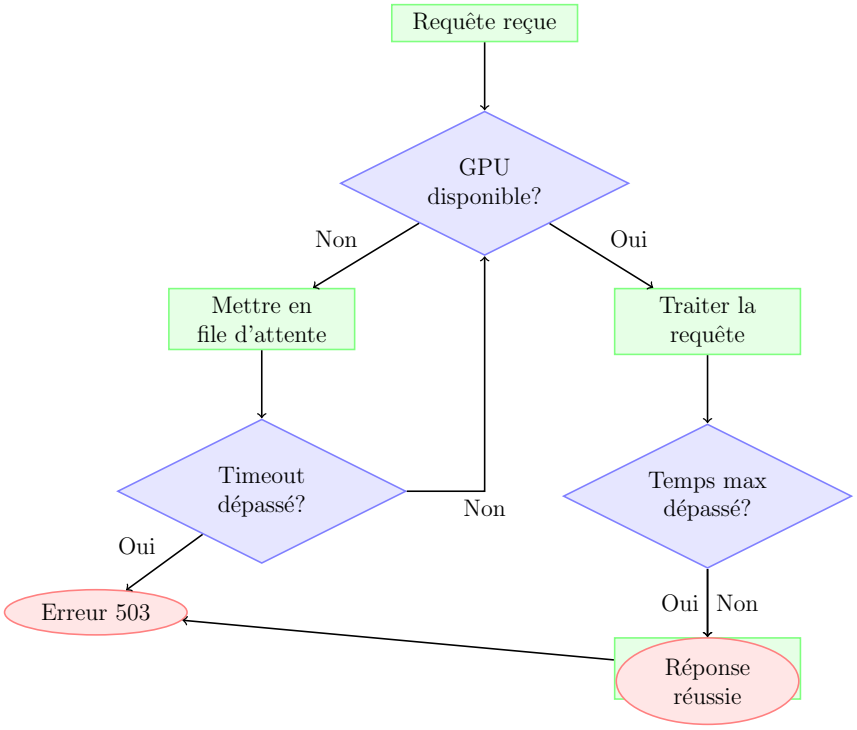

Figure 5: Flowchart de Gestion des Requêtes avec Timeouts

3 Deuxième hypothèse

On a aussi essayé un autre LLM celui de ChatGPT, on a lui demandé de nous expliquer la partie de moteur d'inférence et comment il ne peut pas se bloquer si l'architecture l'utilise. Comme réponse, DeepSeek utilise non seulement le moteur d'inférence mais avec les modèles intelligents. En effet, les modèles se comportent comme "le cerveau" pour générer la réponse à partir de requête de l'utilisateur, et le moteur d'inférence se comporte comme "les muscles" en fournissant la réponse des modèles à l'utilisateur.

4 Conclusion

L'architecture de DeepSeek utilise une approche multi-couches pour éviter les blocages :

1. **Répartition de charge** sur plusieurs workers GPU 2. **Système de files d'attente** avec priorités 3. **Circuit breakers** pour isoler les pannes 4. **Auto-scaling** basé sur les métriques 5. **Timeouts agressifs** pour libérer les ressources

Cette approche garantit que même en cas de pic de charge, le système reste responsive et dégrade gracieusement ses performances plutôt que de bloquer complètement.