Ideo Lab

16th July 2025

Version 2.1

MONITORING
Guillaume Oneill
LockHeed LLC

MONITORING, ROBOTISATION & IA

Table des Matières

📄 Table des Matières – Roadmap IA pour Monitoring PostgreSQL

---

4. Playbook Phase 1 – Instrumentation & Observabilité

4.1. Préparation et cadrage (instances, métriques, tagging)

4.2. Installation extensions PostgreSQL (pg_stat_statements, auto_explain)

4.3. Collecte métriques système (Telegraf, Node Exporter)

4.4. Pipeline de monitoring (Prometheus, Exporters)

4.5. Dashboards initiaux Grafana

4.6. Gestion et centralisation logs (ELK, Loki)

---

5. Playbook Phase 2 – Automatisation Basique

5.1. Système d'alertes (thresholds, alertmanager)

5.2. Scripts d'action automatisée (kill sessions, purge logs, restart)

5.3. Orchestration (Ansible, Cron)

5.4. Alertes corrélées et réduction bruit

5.5. Tests, validation et documentation opérationnelle

---

6. Playbook Phase 3 – Copilot IA (Analytique DBA)

6.1. Préparation data IA (normalisation logs, corpus DBA, RAG)

6.2. Création agent IA (chatbot PostgreSQL connecté aux métriques)

6.3. Analyse continue IA et score santé base

6.4. Génération plans d'actions SQL (index, vacuum, tuning)

6.5. Interface Copilot dans dashboard Grafana

6.6. Sécurité, validation humaine, historique actions IA

7. Playbook Phase 4 – Anticipation & Monitoring Prédictif

7.1. Collecte et structuration données historiques (incidents, métriques)

7.2. Feature engineering pour ML

7.3. Modélisation prédictive (forecast charge, incidents, anomalies)

7.4. Pipeline temps réel (Airflow, Kafka, API ML)

7.5. Dashboard prédictif et alertes préventives

7.6. Automatisation semi-auto préventive

7.7. Boucle d'apprentissage et retrain modèles ML

---

8. Playbook Phase 5 – Self-Healing & Auto-Adaptation

8.1. Moteur de règles intelligentes (policies auto)

8.2. Orchestrateur DB-Healer (scripts SQL, rollback)

8.3. Tuning dynamique en fonction charge

8.4. Auto-indexation adaptative et nettoyage

8.5. Réparation blocages et gestion erreurs

8.6. Scaling automatique (vertical/horizontal)

8.7. IA temps réel pour anomalies nouvelles

8.8. Priorisation actions et exécution autonome

8.9. Mode simulation, sécurité, audit trail

8.10. Dashboard "self-healing" final et KPI

9. Master Plan Consolidé

9.1. Vision globale et objectifs stratégiques

9.2. Roadmap stratégique (tableau Phases 1 → 5)

9.3. Timeline indicative (6 à 12 mois)

9.4. Piliers fonctionnels de transformation (observabilité, IA, proactivité, auto-healing)

9.5. Bénéfices attendus et ROI

9.6. Prochaines étapes immédiates (Mois 1)

Sommaire – Roadmap & Playbooks Monitoring PostgreSQL avec IA

1️⃣ Feuille de route générale (Approche fonctionnelle)

- Vision et objectifs fonctionnels
- Étapes structurantes de la feuille de route (Phase A à F)
- Outils et technologies recommandés
- Bénéfices attendus

2️⃣ Blueprint fonctionnel (Plan directeur)

- Tableau de synthèse par phase (1 à 6)
- Étapes clés à respecter (timeline globale)
- KPI de succès

3️⃣ Playbook Phase 1 & 2 – Instrumentation & Automatisation Basique

- Préparation et cadrage
- Instrumentation PostgreSQL (extensions, métriques, logs)
- Pipeline de monitoring et dashboards initiaux
- Système d'alertes basiques
- Scripts d'action automatisée (kill, purge, restart)
- Tests, validation et documentation

4 Playbook Phase 3 – Copilot IA (Analytique et Assistance DBA)

- Préparation data IA (normalisation, corpus documentaire)
- Création de l'agent IA (chatbot DBA connecté aux métriques et logs)
- Analyse IA continue et score de santé
- Génération de plans d'action SQL automatiques
- Interface Copilot dans dashboard Grafana
- Sécurité, validation humaine et feedback IA

---

5 Playbook Phase 4 – Anticipation et Prédictif

- Collecte et structuration historique des incidents et métriques
- Feature engineering pour ML
- Modélisation prédictive (forecast, classification, anomalies)
- Pipeline temps réel de prédiction
- Dashboard prédictif et actions préventives
- Boucle d'amélioration continue (retrain ML, feedback IA)
- Exemples d'alertes proactives

6 Playbook Phase 5 – Self-Healing & Auto-Adaptation

- Architecture du moteur de règles intelligentes (DB-Healer)
- Tuning dynamique, auto-indexation adaptative
- Réparation blocages et erreurs en temps réel
- Scaling automatique vertical/horizontal
- Boucle d'intelligence continue (IA temps réel, scoring actions)
- Sécurité, mode simulation, supervision et KPI Self-Healing

---

7 Master Plan Consolidé (Stratégie Globale)

- Vision globale et objectifs
- Roadmap stratégique Phases 1 → 5 (tableau)
- Jalons clés et timeline indicative (6-12 mois)
- Piliers fonctionnels de la transformation
- Bénéfices attendus
- Prochaines étapes immédiates (Mois 1)

1️⃣ Vision et Objectifs Fonctionnels

🎯 Objectifs clés

- **Surveillance proactive** des performances et de la santé de la base.
- **Réaction automatisée** en cas d'anomalies ou de risques détectés.
- **Prédiction** des incidents avant qu'ils n'affectent la production.
- **Optimisation continue** sans dépendre d'interventions manuelles répétitives.

🎛️ Valeurs fonctionnelles attendues

- Temps de réponse court face à tout incident.
- Réduction des tâches manuelles répétitives (robotisation).
- Anticipation des goulots d'étranglement (prévenir plutôt que guérir).
- Ajustement automatique (scaling, tuning) en temps réel.

Étapes Structurantes de la Feuille de Route

AGENDA

- Phase A – Foundation (Collecte et Normalisation)
- Phase B – Automatisation et Robotisation Basique
- Phase C – Intelligence Artificielle et Copilot
- Phase D – Anticipation et Prévention
- Phase E – Réglage en Temps Réel (Self-Healing)
- Phase F – Supervision et Gouvernance

2️⃣ Étapes Structurantes de la Feuille de Route

Phase A – Foundation (Collecte et Normalisation)

1. **Instrumentation complète** de la base (pg_stat_statements, pg_stat_activity, logs, métriques système).
2. **Agent d'observation** (Prometheus, Telegraf, ou équivalent) pour collecter CPU, RAM, I/O, locks, temps de requête, index inutilisés.
3. **Pipeline d'événements** centralisé (ex: Kafka ou queue interne) pour unifier toutes les données du monitoring.

Objectif fonctionnel : Disposer d'une base de données de télémétrie fiable, exploitable par une IA ou des règles automatisées.

Phase B – Automatisation et Robotisation Basique

1. **Alerting automatique** (thresholds) sur métriques critiques : latence, connexions bloquées, espace disque.
2. **Actions correctives automatisées simples** :
   - Redémarrage d'un worker bloqué.
   - Kill d'une requête en deadlock.
   - Purge temporaire des fichiers de logs.

Objectif fonctionnel : Première couche de "réaction robotisée" immédiate sans attendre

Phase C – Intelligence Artificielle et Copilot

1. **Moteur d'analyse IA (NLP + ML)** connecté à la télémétrie :
   - Classification des anomalies (IO bound, lock contention, query inefficace).
   - Recommandations automatiques d'indexation ou tuning.
   - Détection d'"early signs" de saturation (forecast CPU, croissance table).
2. **Copilot DBA** :
   - Chatbot ou assistant (via API OpenAI) capable de répondre : "*Pourquoi la base ralentit depuis 10h ?*", "*Que dois-je optimiser ?*"
   - Génération automatique de plans d'actions SQL (création index, vacuum, réorganisation partitions).
3. **Scoring proactif** : chaque minute, une note de "santé" de la base est calculée.

Phase D – Anticipation et Prévention

1. **Modèles prédictifs** (machine learning supervisé) entraînés sur historique d'incidents.
2. **Planification préventive** :
   - Estimation des saturations disque/mémoire dans X heures/jours.
   - Alerte "préventive" avec suggestion d'actions avant impact réel.
3. **Test automatisé d'optimisations** en environnement miroir, déploiement validé en production si bénéfique.

Phase E – Réglage en Temps Réel (Self-Healing)

1. **Règles dynamiques IA** :
   - Ajustement du pool de connexions selon la charge.
   - Activation/désactivation automatique d'index en fonction du workload.
   - Lancement de vacuum/analyze à des moments optimaux détectés par l'IA.
2. **Autoscaling** base ou répartition trafic (si cluster/PG Bouncer/Patroni) déclenchés par l'IA.

Phase F – Supervision et Gouvernance

1. **Dashboard intelligent** (Grafana + AI) affichant :
   - État temps réel, tendances prédites, recommandations IA.
   - Historique des actions automatiques appliquées.
2. **Contrôle humain assisté** : l'IA propose, le DBA valide ou laisse l'automatisation agir seule.
3. **Boucle de feedback** : IA apprend de chaque incident ou ajustement manuel.

3️⃣ Outils et Technologies Recommandés

- **Collecte & Monitoring** : Prometheus, pg_stat_statements, ELK Stack.
- **Automatisation** : Ansible, scripts Python avec API PostgreSQL, triggers automatisés.
- **IA / Copilot** :
  - API OpenAI pour recommandations NLP.
  - Modèles ML pour prédiction (TensorFlow, PyTorch, Prophet).
  - Agent RAG (Recherche augmentée par IA) sur la documentation interne + logs.
- **Interface** : Grafana + plugin Copilot intégré (chat + auto-actions).

4️⃣ Bénéfices Attendus

- Réduction drastique du temps de résolution incidents.
- Moins de "surprises" (anticipation des pannes ou lenteurs).
- Amélioration continue sans surcharge du DBA.
- Passage d'un mode réactif → **mode proactif & auto-adaptatif.**

Blueprint – Automatisation IA du Monitoring PostgreSQL

Blueprint – Automatisation IA du Monitoring PostgreSQL

Phase	Objectifs Fonctionnels	Livrables	Rôles impliqués	Outils/Techno
1 **Fondation – Observabilité**	- Centraliser toutes les métriques pertinentes. - Garantir une visibilité complète sur l'état de PostgreSQL (SQL, système, réseau).	- Schéma d'instrumentation (pg_stat_statements, logs, métriques OS). - Pipeline Prometheus/Telegraf → Time-series DB. - Dashboards Grafana initiaux.	DBA, DevOps	PostgreSQL, Prometheus, Telegraf, Grafana
2 **Automatisation Basique (Réactivité)**	- Détecter et réagir instantanément aux incidents connus. - Supprimer les tâches manuelles répétitives.	- Système d'alertes (thresholds CPU, locks, deadlocks). - Scripts Python/Ansible pour actions correctives (kill query, restart worker, purge logs).	DBA, SRE	Ansible, Python, pgAdmin API
3 **IA Copilot (Analytique & Conseils)**	- Comprendre les causes racines via l'IA. - Générer recommandations SQL optimisées (index, vacuum, query tuning). - Chatbot d'assistance DBA.	- Agent Copilot connecté à la télémétrie. - API OpenAI (prompt SQL tuning, diagnostic). - Tableau "Santé Base" avec score AI.	DBA, Data Scientist	API OpenAI, GPT-4o, Python, RAG sur logs & doc

4 **Anticipation (Prédictif)**	- Prévoir les risques avant qu'ils n'impactent la prod. - Planifier préventivement les actions correctrices.	- Modèles ML prédictifs (forecast saturation disque/mémoire, croissance table). - Alertes "préventives" avec horizon temps. - Rapport préventif quotidien/hebdo IA.	Data Scientist, SRE	TensorFlow/PyTorch, Prophet, Pandas
5 **Réglages Temps Réel (Self-Healing)**	- Ajuster automatiquement les paramètres selon la charge. - Réparer sans intervention humaine quand possible.	- Règles IA pour scaling pool connexions. - Vacuum/analyze déclenché par IA. - Auto-création ou suppression d'index selon usage.	DBA, SRE	Python operator, PL/pgSQL, Patroni, AI policy engine
6 **Gouvernance & Feedback**	- Superviser toutes les actions IA/auto. - Améliorer en continu la fiabilité des prédictions.	- Dashboard final "proactif" (temps réel + prévisions + actions auto appliquées). - Historique et audit trail IA. - Processus de validation humaine (semi-auto mode).	CTO, DBA, Product Owner	Grafana, Tableau BI, Workflow Jira/Confluence

🎯 Étapes Clés à Respecter

1. **Phase pilote** (Base de test) : instrumentation + premiers scripts auto (1-2 mois).
2. **Déploiement en prod** avec automatisations simples + Copilot en mode "conseil uniquement" (2-3 mois).
3. **Activation IA proactive** (actions auto validées puis non validées) sur incidents simples (3-5 mois).
4. **Apprentissage continu** : enrichissement modèles ML, tuning des règles IA, feedback DBA intégré (6-9 mois).
5. **Mode Full Auto-Adaptive** : tuning dynamique, scaling auto, auto-indexation (9-12 mois).

💡 KPI à suivre pour mesurer le succès

- Taux de détection d'incidents avant impact réel (>80%).
- Temps moyen de résolution réduit de 70% (MTTR).

Playbook Opérationnel – Phases 1,2,3 & 4

AGENDA PHASES OPERATIONELLES

- PHASE 1 – Instrumentation & Observabilité (Durée : 2-4 semaines)
- PHASE 2 – Automatisation Basique & Réactivité (Durée : 3-5 semaines)
- Playbook Opérationnel – Phase 3 (Copilot IA)
- Playbook Opérationnel – Phase 4 (Anticipation & Prédictif)

🎯 Objectif global

- **Phase 1 (Fondation)** : Mettre en place une base d'observabilité complète (collecte fiable de données).
- **Phase 2 (Automatisation basique)** : Réagir automatiquement aux incidents simples et récurrents.

📌 PHASE 1 – Instrumentation & Observabilité (Durée : 2-4 semaines)

1️⃣ Préparation et cadrage

- [ ] Identifier toutes les instances PostgreSQL à monitorer (prod, pré-prod, réplica).
- [ ] Lister les métriques critiques à suivre :
  - Requêtes lentes, locks, CPU, I/O, connexions actives, taille tables/index.
- [ ] Définir un schéma d'étiquetage (tags) pour différencier environnements/services.

2️⃣ Installation des extensions PostgreSQL

- [ ] Activer **pg_stat_statements** pour collecter l'historique des requêtes lentes.
- [ ] Installer **pg_stat_activity** pour suivre les sessions et verrous en temps réel.
- [ ] Configurer **auto_explain** pour capturer plans d'exécution des requêtes lourdes.

3️⃣ Collecte métriques système

- [ ] Déployer **Telegraf** ou **Node Exporter** sur chaque serveur DB.
- [ ] Configurer collecte : CPU, RAM, swap, I/O disque, latence réseau.

4️⃣ Pipeline de monitoring

- [ ] Installer **Prometheus** (ou VictoriaMetrics) pour collecter les métriques PostgreSQL.
- [ ] Connecter exporter PostgreSQL (`postgres_exporter`) aux instances.
- [ ] Définir règles de rétention (ex: 30 jours sur métriques détaillées).

5️⃣ Dashboards initiaux

- [ ] Créer Dashboard Grafana “Vue DBA” : requêtes lentes, deadlocks, erreurs SQL.
- [ ] Créer Dashboard “Vue Ops” : CPU, I/O, espace disque, load average.
- [ ] Mettre en place une page “Health Check Global” (une tuile par instance).

6️⃣ Log Management

- [ ] Centraliser logs PostgreSQL via ELK (ElasticSearch + Logstash + Kibana) ou Loki.
- [ ] Activer logs d’erreur, checkpoints, verrous, autovacuum, connexions refusées.
- [ ] Normaliser format (JSON si possible) pour exploitation IA ultérieure.

PHASE 2 – Automatisation Basique & Réactivité (Durée : 3-5 semaines)

7️⃣ Système d'alertes

- [ ] Définir seuils critiques :
  - Connexions > 80% du max_connections
  - Requêtes > 5s
  - Deadlocks > 1/minute
  - Espace disque < 10% libre
- [ ] Créer alertes Prometheus → AlertManager → Slack/Email.

8️⃣ Scripts d'action automatisée

- [ ] Script Python `kill_blocking_sessions.py` : termine les sessions bloquées > X minutes.
- [ ] Script `cleanup_logs.sh` : purge logs anciens > 7 jours en cas de disque plein.
- [ ] Script `restart_pg_worker.sh` : redémarre service si crash détecté.

9️⃣ Orchestration et planification

- [ ] Mettre scripts dans **Ansible playbooks** ou **Cron jobs supervisés**.
- [ ] Définir règles simples (ex: max 1 kill/10min pour éviter effet ping-pong).
- [ ] Journaliser toute action auto dans un fichier central (historique).

🔟 Alertes intelligentes simples

- [ ] Activer **alerte corrélée** : si CPU élevé + queries lentes → signal critique unique.
- [ ] Marquer alertes "répétitives" comme à faible bruit (éviter spam DBA).
- [ ] Définir workflow "humain alerté après 3 auto-actions en 30 min".

1️⃣1️⃣ Tests et validation

- [ ] Simuler blocage d'une table → vérifier kill auto OK.
- [ ] Simuler espace disque saturé → purge logs fonctionne.
- [ ] Valider qu'aucune action auto ne perturbe prod (dry-run au début).

1️⃣2️⃣ Documentation et transfert

- [ ] Rédiger "Runbook automatisé" (procédures, scripts, seuils).
- [ ] Former l'équipe DBA à l'usage des dashboards et scripts.
- [ ] Mettre en place une checklist hebdo de validation (santé automatisation).

📌 Livrables finaux Phase 1-2

- ✅ Dashboards Grafana + Prometheus connectés.
- ✅ Scripts automatisés prêts en production (kill, purge,
- ✅ Alertes Slack/Email fonctionnelles et testées.
- ✅ Historique d'actions automatisées consultable.
- ✅ Documentation claire "Auto-monitoring PostgreSQL

Phase 3 - Playbook Opérationnel – (Copilot IA)

AGENDA - Playbook Opérationnel

- ÉTAPE 1 – Préparation Data IA (Durée : 2-3 semaines)
- ÉTAPE 2 – Copilot IA (Durée : 3-5 semaines)
- ÉTAPE 3 – Interface Copilot & Interaction (Durée : 2-3 semaines)
- ÉTAPE 4 – Validation et Amélioration (Durée : 2-4 semaines)

🎯 Objectif global

- Exploiter l'IA pour **analyser, diagnostiquer, conseiller et prédire** les problèmes PostgreSQL.
- Fournir un **assistant conversationnel DBA** capable de répondre, expliquer et recommander.
- Poser les bases du futur mode **proactif et auto-adaptatif** (Phase 4-5).

📌 ÉTAPE 1 – Préparation Data IA (Durée : 2-3 semaines)

1️⃣ Normalisation des données collectées

- ☐ Structurer les logs PostgreSQL (JSON) pour ingestion IA.
- ☐ Indexer les logs et métriques dans une base de recherche (ElasticSearch, OpenSearch).
- ☐ Créer un schéma unifié (timestamp, instance, type_event, query, temps_exec, lock, utilisateur, statut).

2️⃣ Historisation longue durée

- ☐ Étendre rétention des données à 6-12 mois pour apprentissage ML.
- ☐ Stocker événements d'incidents (deadlocks, OOM, saturation disque).
- ☐ Marquer incidents résolus pour apprentissage supervisé ultérieur.

3️⃣ Corpus documentaire DBA

- ☐ Collecter doc PostgreSQL interne (playbooks DBA, tuning SQL, indexation).
- ☐ Inclure FAQ DBA, guides d'optimisation, règles de bonnes pratiques.
- ☐ Stocker en base RAG (vecteurs) pour recherche sémantique IA.

📌 ÉTAPE 2 – Copilot IA (Durée : 3-5 semaines)

4️⃣ Création de l'agent IA (Chatbot DBA)

- ☐ Déployer un **service Python** interrogeant API OpenAI (GPT-4o ou équivalent).
- ☐ Définir contexte : "Tu es un DBA expert PostgreSQL. Tu expliques, diagnostiques, proposes SQL correctif."
- ☐ Connecter l'agent à :
  - (a) **Logs** → accès en lecture via API ElasticSearch
  - (b) **Métriques Prometheus** → snapshot live
  - (c) **Corpus RAG** → réponses basées sur doc interne

Exemples de questions possibles :

- "Pourquoi le CPU a explosé entre 14h-15h sur l'instance A ?"
- "Donne-moi les 5 requêtes les plus lentes depuis 24h."
- "Quel index manque sur la table orders ?"
- "Propose un plan d'optimisation des connexions."

5️⃣ Analyse IA en continu

- [ ] Créer **scripts cron** ou service continu analysant :
  - Requêtes lentes répétitives → recommandation index.
  - Deadlocks récurrents → suggestion de réécriture transactionnelle.
  - Croissance anormale taille table → plan de partitionnement.
- [ ] Générer un **score de santé IA** (0-100) basé sur latence, erreurs, contention, risque saturation.

6️⃣ Génération de plans d'actions SQL automatiques

- [ ] Prompts IA préconstruits pour :
  - Création index manquant (`CREATE INDEX ...`).
  - Analyse vacuum/analyze adapté.
  - Suggestion réécriture requête lente (`EXPLAIN -> alternative`).
- [ ] Scripts Python pour **tester en environnement staging** avant exécution en prod.

📌 ÉTAPE 3 – Interface Copilot & Interaction (Durée : 2-3 semaines)

7️⃣ Intégration dans le Dashboard

- ☐ Ajouter un **onglet "Copilot DBA"** dans Grafana ou UI custom :
  - Chat interactif (questions → réponses IA).
  - Bouton “Proposer correctif” → plan SQL détaillé.
  - Vue "santé IA" avec recommandations classées (urgent / important / optionnel).

8️⃣ Gestion de la sécurité

- ☐ Mode “read-only” par défaut (aucune action directe IA sur prod).
- ☐ Validation humaine obligatoire avant exécution de correctifs.
- ☐ Historique de toutes les requêtes IA et actions proposées.

📌 ÉTAPE 4 – Validation et Amélioration (Durée : 2-4 semaines)

9️⃣ Tests utilisateurs

- ☐ Scénarios :
  - Incident simulé → IA doit identifier cause + solution.
  - Query lourde → IA doit suggérer index correct.
  - Saturation mémoire → IA doit recommander param tuning.
- ☐ Mesurer :
  - Temps gagné par rapport à diagnostic manuel.
  - Pertinence des suggestions (>80% validées).

🔟 Enrichissement du modèle

- ☐ Ajouter feedback DBA (👍/👎) sur réponses IA pour affiner prompts et RAG.
- ☐ Envisager modèle local finement ajusté (fine-tuning) avec logs internes.

📌 Livrables Phase 3

- ✅ Service Copilot IA déployé (API REST ou chatbot web).
- ✅ Recherche sémantique sur logs & doc DBA.
- ✅ Dashboard intégré avec onglet IA.
- ✅ Plans SQL correctifs générés automatiquement.
- ✅ Score de santé IA affiché en temps réel.
- ✅ Documentation “Copilot DBA – Usage & Bonnes pratiques”.

📌 Livrables Phase 3

- ✅ Service Copilot IA déployé (API REST ou chatbot web).
- ✅ Recherche sémantique sur logs & doc DBA.
- ✅ Dashboard intégré avec onglet IA.
- ✅ Plans SQL correctifs générés automatiquement.
- ✅ Score de santé IA affiché en temps réel.
- ✅ Documentation "Copilot DBA – Usage & Bonnes pratiques".

Sample use of OpenAI or Gemini « Prompt »

💡 Exemples de Prompts préconfigurés (fonctionnels)

- **Diagnostic incident**

  "Analyse les logs PostgreSQL des dernières 2h et donne-moi les 3 causes probables de l'augmentation de la latence (>1s)."

- **Requête lente**

  "Cette requête SELECT ... prend 8s. Donne-moi 2 alternatives plus rapides basées sur l'indexation ou la réécriture SQL."

- **Tuning automatique**

  "Analyse les paramètres actuels (work_mem, shared_buffers, max_connections) et propose un ajustement optimal pour 200 connexions simultanées."

- **Plan préventif**

  "Identifie les 5 tables qui risquent de dépasser 100M lignes dans 3 mois et propose une stratégie de partitionnement ou indexation."

Playbook Opérationnel – Phase 4 (Anticipation & Prédictif)

🎯 Objectif global

- **Prévoir** les incidents (saturation, blocage, explosion du temps de réponse, crash potentiel) à court, moyen et long terme.
- **Prioriser les actions préventives** (indexation, nettoyage, scaling) avant impact réel.
- Fournir un **tableau de bord prédictif** alimenté par l'IA pour guider les décisions DBA/SRE.

Agenda phase Opérationelle 4

- ÉTAPE 1 – Collecte & Structuration des Données Historiques (Durée : 2-3 semaines)
- ÉTAPE 2 – Modélisation Prédictive (Durée : 4-6 semaines)
- ÉTAPE 3 – Pipeline de Prédiction Temps Réel (Durée : 3-5 semaines)
- ÉTAPE 4 – Dashboard Prédictif & Actions Préventives (Durée : 2-4 semaines)
- ÉTAPE 5 – Amélioration continue & Feedback (Durée : continu)

📌 ÉTAPE 1 – Collecte & Structuration des Données Historiques (Durée : 2-3 semaines)

1️⃣ Extension de l'observabilité

- ☐ Récupérer **12-24 mois de données** (ou minimum 3-6 mois) :
  - Métriques techniques (CPU, I/O, connexions, requêtes lentes).
  - Logs d'incidents (deadlocks, crash, OOM).
  - Croissance taille tables/index.
  - Paramètres PostgreSQL modifiés au fil du temps.
- ☐ Stocker données dans un **Data Lake interne** (PostgreSQL ou TSDB dédiée).

2️⃣ Feature engineering pour ML

- ☐ Créer features temps-série :
  - Moyenne mobile temps de réponse, variance latence.
  - Ratio lectures/écritures, taux d'index scans.
  - Taux de croissance table/jour, fragmentation index.
  - Nombre d'alertes/seuils franchis.
- ☐ Ajouter **context features** :
  - Nombre d'utilisateurs connectés.
  - Version PostgreSQL, réglages mémoire.
  - Déploiements applicatifs liés à incidents.

📌 ÉTAPE 2 – Modélisation Prédictive (Durée : 4-6 semaines)

3️⃣ Sélection des modèles

- ☐ **Forecast charge & saturation :**
  - Prophet ou ARIMA (prédiction CPU, I/O, connexions).
- ☐ **Prédiction incidents :**
  - Random Forest / XGBoost (classification risque incident).
  - Réseaux neuronaux simples pour détection anomalies complexes.
- ☐ **Anomalie comportementale :**
  - Isolation Forest, Auto-encoders pour détecter pattern "jamais vu".

4️⃣ Entraînement et validation

- ☐ Découper dataset : 70% entraînement, 20% validation, 10% test.
- ☐ Évaluer :
  - Précision détection incidents futurs (>85% visée).
  - Anticipation saturation (prévision >90% avant 2h minimum).
- ☐ Ajuster features et hyperparamètres.

📌 ÉTAPE 3 – Pipeline de Prédiction Temps Réel (Durée : 3-5 semaines)

5️⃣ Architecture du pipeline

- ☐ Job batch/streaming (Airflow ou Kafka Streams) :
  - Collecte métriques chaque minute.
  - Envoi features → modèle ML → scoring en quasi temps réel.
- ☐ Stockage prédictions dans **table "risques prévisionnels"** (TSDB ou DB interne).
- ☐ Génération d'**alertes proactives** :
  - Exemple : "Probabilité 85% saturation disque dans 5h sur instance prod-2."
  - Exemple : "Risque élevé deadlocks dans 24h lié à croissance table X."

📌 ÉTAPE 4 – Dashboard Prédictif & Actions Préventives (Durée : 2-4 semaines)

6️⃣ Tableau de bord avancé

- ☐ Vue **Prévision charge** (CPU, I/O, connexions) avec horizon 1h, 24h, 7j.
- ☐ Vue **Risque incidents** (heatmap par instance, probabilité %).
- ☐ Liste **actions préventives suggérées** :
  - Nettoyer tables inactives avant saturation disque.
  - Ajouter index anticipé sur colonne fortement sollicitée.
  - Prévoir scaling cluster avant pic charge (vendredi soir par ex.).

7️⃣ Automatisation semi-auto

- ☐ Connecter pipeline prédictif → moteur d'orchestration :
  - Générer tickets automatiques (Jira) pour DBA.
  - Déclencher scripts préventifs planifiés (ex: VACUUM FULL avant saturation).
- ☐ Mode test : validation humaine obligatoire avant action auto.

📌 ÉTAPE 5 – Amélioration continue & Feedback (Durée : continu)

8️⃣ Boucle d'apprentissage

- ☐ Chaque incident futur corrigé enrichit dataset (étiquetage post-mortem).
- ☐ Copilot IA (Phase 3) explique en langage naturel :

  "Risque élevé de deadlock demain, cause probable : accumulation de verrous table orders."
- ☐ Retrain modèle ML chaque mois (auto-pipeline CI/CD ML).

📌 Livrables Phase 4

- ✅ Dataset complet incidents + métriques enrichis.
- ✅ Modèles ML validés (forecast & classification incidents).
- ✅ API de scoring temps réel branchée sur monitoring existant.
- ✅ Tableau de bord prédictif (Grafana/Streamlit) avec horizon temps.
- ✅ Workflow préventif semi-auto validé.
- ✅ Documentation "Monitoring prédictif PostgreSQL – Niveau Proactif".

💡 Exemples d'Alertes Proactives générées

- ⏳ *"Probabilité 92% saturation tablespace* `pg_default` *dans 7h. Actions recommandées : purge logs + extension disque + archiving."*
- ⚠️ *"Taux de croissance table* `transactions` *→ +5M lignes/jour. Risque 80% de requêtes lentes <72h sans index composite sur (user_id, date)."*
- 🧠 *"Pattern similaire à incident du 2025-05-14 détecté. Risque 68% de deadlock sur* `orders` *demain."*

Playbook Opérationnel – Phase 5 (Self-Healing & Auto-Adaptation)

🎯 Objectif global

- Passer d'une posture **proactive** à une posture **auto-adaptative**, où la plateforme :
  - **Analyse** en continu l'état de PostgreSQL.
  - **Prend des décisions en temps réel** (scaling, tuning, correctifs).
  - **Applique automatiquement** des actions correctrices validées par l'IA.
  - Réduit au **minimum les incidents critiques**, même en l'absence de DBA.

AGENDA - Playbook Opérationnel – Phase 5

- ÉTAPE 1 – Architecture Self-Healing (Durée : 2-3 semaines)
- ÉTAPE 2 – Actions de correction automatisées (Durée : 4-6 semaines)
- ÉTAPE 3 – Boucle d’intelligence continue (Durée : 3-5 semaines)
- ÉTAPE 4 – Sécurité et Gouvernance

📌 ÉTAPE 1 – Architecture Self-Healing (Durée : 2-3 semaines)

1️⃣ Moteur de règles intelligentes

- ☐ Définir une base de **règles automatiques** (policies) :
  - Exemple : "*Si connexions > 90% → augmenter pool max de 20%.*"
  - Exemple : "*Si deadlock détecté et bloque requête > 3min → kill session fautive.*"
- ☐ Prioriser actions **safe** (non destructives) en exécution auto, ex:
  - VACUUM, analyse, kill session, scaling vertical/horizontal.
  - Pas de drop index/alter table sans validation humaine.

2️⃣ Orchestrateur d'actions

- ☐ Déployer un **service Python/Go** ("DB-Healer") interfacé :
  - Monitoring → Moteur règles → Exécution SQL ou script.
  - Journalisation de chaque action (audit complet).
- ☐ Support rollback en cas d'action auto problématique (log + revert).

📌 ÉTAPE 2 – Actions de correction automatisées (Durée : 4-6 semaines)

3️⃣ Tuning dynamique

- ☐ Ajustement automatique des paramètres PostgreSQL selon charge :
  - `work_mem`, `max_connections`, `autovacuum` threshold.
  - Basé sur règles + recommandations IA (Phase 3).
- ☐ Redémarrage ou reload dynamique si nécessaire (sans downtime si possible).

4️⃣ Auto-indexation adaptative

- ☐ Détection répétition de requêtes lentes → création automatique index sur staging.
- ☐ Test performance → si gain mesuré > seuil (ex 40%), déploiement auto en prod.
- ☐ Suppression index inutilisé > 90j après validation IA.

5️⃣ Réparation blocages & erreurs

- ☐ Kill auto sessions en deadlock ou bloquant autres requêtes > X min.
- ☐ Redémarrage automatique process `autovacuum` si bloqué.
- ☐ Rotation logs ou archive tables en cas d'espace disque < 5%.

6️⃣ Scaling automatique (cluster ou instance unique)

- ☐ Si métriques charge > seuil critique répété :
  - Déclenchement auto **scaling vertical** (RAM, CPU).
  - Ou **scaling horizontal** (ajout read replica).
- ☐ Balanceur de charge (PgBouncer/HAProxy) mis à jour automatiquement.

📌 ÉTAPE 3 – Boucle d'intelligence continue (Durée : 3-5 semaines)

7️⃣ Détection comportement anormal (IA temps réel)

- ☐ L'IA analyse flux métriques/logs en direct :
  - Détection anomalies non couvertes par règles.
  - Proposition d'action immédiate ou blocage préventif.
- ☐ Exemple :
  - *"Nouvelle requête consomme 95% CPU → mise en quarantaine automatique et alerte DBA."*

8️⃣ Priorisation automatique des actions

- ☐ Système de scoring impact/risque :
  - Score urgence (blocage majeur vs optim minime).
  - Score fiabilité action (probabilité succès basé historique).
- ☐ Actions à score > 80% appliquées **directement**, sinon attente validation humaine.

9️⃣ Feedback IA → évolution des règles

- ☐ Après chaque incident traité :
  - L'IA compare actions exécutées vs alternatives possibles.
  - Amélioration automatique des policies ("policy learning").
- ☐ Retrain modèles prédictifs (Phase 4) en intégrant incidents récents.

📌 ÉTAPE 4 – Sécurité et Gouvernance

🔟 Mode sécurité

- ☐ Actions auto exécutées en mode "**simulation**" 1 mois avant activation réelle.
- ☐ Validation humaine obligatoire sur actions "risquées" (drop index, partitionnement).
- ☐ Journal complet envoyé chaque jour aux DBA (audit trail).

1️⃣1️⃣ Supervision globale

- ☐ Dashboard final "**Self-Healing PostgreSQL**" :
  - Actions automatiques effectuées (timestamp, gain).
  - Problèmes corrigés sans intervention humaine.
  - Statistiques : temps moyen résolution, incidents évités.

📌 Livrables Phase 5

- ✅ Moteur de règles IA + orchestrateur DB-Healer.
- ✅ Scripts d'autocorrection validés (tuning, kill, vacuum, indexation).
- ✅ Scaling automatique fonctionnel sur pics de charge.
- ✅ Mode simulation → mode réel sécurisé.
- ✅ Dashboard "self-healing" temps réel.
- ✅ Documentation "Politique d'auto-réparation PostgreSQL".

💡 Exemple d'actions automatiques typiques

- 🔄 *"Augmentation auto de work_mem de 64MB à 128MB (charge élevée détectée) → gain latence -35%."*
- 🧹 *"VACUUM auto déclenché sur table orders (bloat 35%)."*
- ⚠️ *"Session PID 1234 tuée (bloque 12 requêtes depuis 5 min)."*
- 🚀 *"Read replica ajouté (charge prévisionnelle 120% dans 2h)."*
- ✅ *"Index auto créé sur users(last_login) après 24h tests validés → temps requête -70%."*

KPI de réussite Self-Healing

- ≥ 70% incidents corrigés **sans intervention humaine.**
- MTTR (temps moyen résolution) divisé par 5.
- Nombre d'alertes critiques envoyées à DBA < 20% de la situation initiale.
- Aucune action auto causant régression majeure (<1% cas).

Master Plan – Automatisation & IA pour le Monitoring PostgreSQL

1️⃣ Vision Globale

🎯 Objectif

Mettre en place une plateforme **autonome**, capable de :

- **Surveiller** en continu la santé PostgreSQL (observabilité complète).
- **Réagir automatiquement** aux incidents simples (self-repair niveau 1).
- **Diagnostiquer & recommander** via un **Copilot IA** expert DBA.
- **Prédire les incidents** avant qu'ils ne surviennent (proactivité).
- **S'auto-ajuster** et corriger en temps réel (mode self-healing complet).

Roadmap Stratégique

Phase	Nom	Durée estimée	Objectifs clés	Livrables principaux	KPI de succès
1	Fondation (Observabilité)	2-4 semaines	Collecte complète métriques et logs PostgreSQL	- Exporters + Prometheus - Dashboards Grafana - Logs centralisés	100% instances couvertes Visibilité en temps réel
2	Automatisation Basique	3-5 semaines	Réaction auto aux incidents récurrents (scripts correctifs)	- Alerting thresholds - Scripts kill/purge/restart - Historique actions auto	MTTR ↓ 30-50% ≥50% incidents mineurs auto-résolus
3	Copilot IA (Analytique)	5-8 semaines	Assistance IA : diagnostic, tuning SQL, recommandations	- Service Copilot (chat) - Score santé IA - Plans d'optimisation SQL auto-générés	Pertinence IA > 80% Temps diagnostic ↓ 50%
4	Anticipation (Prédictif)	6-10 semaines	Détection proactive risques incidents et saturation	- Modèles ML forecast & classification - Tableau prédictif risques - Alertes préventives	≥ 70% incidents prévus Alerte ≥ 2h avant impact
5	Self-Healing (Auto-adaptatif)	8-12 semaines	Réglage temps réel, actions auto-validées, scaling dynamique	- Moteur règles IA - Orchestrateur DB-Healer - Dashboard self-healing	≥ 70% incidents corrigés sans humain MTTR ÷ 5 Alertes critiques -80%

3️⃣ Jalons Clés (Timeline indicative 6-12 mois)

- **Mois 1** → Phase 1 (Observabilité complète).
- **Mois 2-3** → Phase 2 (Scripts & automatisation simple).
- **Mois 3-5** → Phase 3 (Copilot IA en mode "conseil").
- **Mois 5-7** → Phase 4 (Prédictions incidents).
- **Mois 7-10** → Phase 5 (Auto-adaptation + correctifs sans humain).
- **Mois 11-12** → Stabilisation, tuning, passage en mode **DB Autonome**.

4️⃣ Pilier Fonctionnels du Master Plan

- 🔍 **Observabilité renforcée** : instrumentation exhaustive, logs, dashboards.
- 🤖 **Robotisation** : scripts automatiques pour incidents connus.
- 🧠 **Intelligence artificielle** : diagnostic, tuning, recommandations SQL.
- 📈 **Prédictif** : anticiper saturations, anomalies, croissance anormale.
- 🔄 **Auto-healing** : tuning dynamique, scaling, autocorrection intelligente.
- 🔒 **Gouvernance** : journalisation, sécurité, validation humaine contrôlée.

---

5 Bénéfices attendus

- **Disponibilité accrue** de PostgreSQL (incidents évités ou corrigés avant impact).
- **Réduction drastique du MTTR** et des interventions manuelles.
- **Vision prédictive** des risques à court/moyen terme.
- **Optimisation continue** des performances sans surcharge DBA.
- **Base résiliente & auto-adaptative**, modèle inspiré du SRE (Site Reliability Engineering).

---

6 Prochaines étapes immédiates (Mois 1)

1. **Valider le périmètre** : instances PostgreSQL à couvrir, SLA, objectifs de disponibilité.
2. **Mettre en place Phase 1 (Instrumentation)** : Prometheus, Grafana, centralisation logs.
3. **Former l'équipe DBA/DevOps** à l'usage des dashboards et alertes.
4. **Planifier Phase 2 (scripts auto)** et définir seuils critiques avec experts.
5. **Préparer datasets historiques** pour IA (Phase 3-4).

MONITORING

📁 Structure Générale des Fiches Techniques : "*Monitoring*"

Chaque fiche suivra le format standard suivant (personnalisable) :

- Titre
- Catégorie / Thématique
- Description générale
- Problèmes résolus
- Architecture / Fonctionnement
- Bonnes pratiques
- Exemples concrets
- Commandes / API / Scripts
- Outils associés
- Risques / Limitations
- Tags / Niveau

🧭 Thématiques Principales pour les Fiches "Monitoring"

1. Introduction au Monitoring

- Définitions : Monitoring, Observability, Logging, Alerting, Tracing
- Pourquoi monitorer ? (Objectifs : disponibilité, performance, prédiction de panne)
- Systèmes vs applications vs réseau
- KPI clés : SLA, Uptime, Latence, Erreurs, Saturation

2. Types de Monitoring

Type	Objectifs	Exemples
Monitoring Système	CPU, RAM, Disque, Processus	`top`, `htop`, `vmstat`
Monitoring Réseau	Bande passante, latence, perte de paquets	`ping`, `traceroute`, `netstat`, `iftop`
Monitoring Applicatif	Health check, erreurs, métriques métier	Middleware, frameworks
Monitoring de Base de Données	Temps de réponse, verrous, requêtes lentes	`pg_stat_activity`, `EXPLAIN ANALYZE`
Monitoring des Containers	Ressources par conteneur, logs	Docker Stats, cAdvisor
Monitoring Cloud / Infra-as-Code	AWS, GCP, Azure, Terraform	CloudWatch, StackDriver

3. Méthodologies & Concepts

- Le modèle RED (Rate, Error, Duration)
- Le modèle USE (Utilization, Saturation, Errors)
- Alerting vs Thresholding vs Anomaly Detection
- Taux d'échantillonnage et agrégation (histogrammes, heatmaps, percentiles)
- Correlation Logs / Metrics / Traces
- Centralisation vs décentralisation des données
- Exporters, agents, agents sidecar

PRINCIPAUX OUTILS DE MONITORING

- DataDog
- Prometeus
- Grafana
- Zabbix
- Centreon
- Elastic Stack (ELK/EFK)
- AWS CloudWatch

🔹 A. Datadog

- **Description** : SaaS complet metrics + logs + APM
- **Avantages** : Visualisation temps réel, alerting avancé, intégrations
- **Fonctionnement** : Agent → API → Dashboard
- **Exemples** :

bash Copier Modifier

```
datadog-agent status
datadog-ci synthetics run-tests
```

- **Bonnes pratiques** : tags, auto-instrumentation APM, budgets de logs
- **Intégrations** : AWS, Kubernetes, PostgreSQL...

🔹 B. Prometheus + Grafana

- **Prometheus** : collecte via *pull*, stocke en TSDB
- **Grafana** : visualisation + alerting
- **Stack typique** : Prometheus → Alertmanager → Grafana
- **Exemples** :

yaml Copier Modifier

```yaml
scrape_configs:
  - job_name: 'node'
    static_configs:
      - targets: ['localhost:9100']
```

- **Exporter communs** : node_exporter, postgres_exporter

🔹 C. Zabbix / Centreon

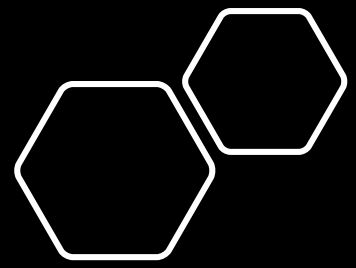
- Solution legacy / on-premise
- Utilisation en environnement réglementé (finance, secteur public)
- Complexité de configuration
- Surveillance fine des hôtes (SNMP, trap, proxy)

🔹 D. Elastic Stack (ELK / EFK)

- Logs et events → Elasticsearch
- **Beats** / **Logstash** / **Fluentd** pour ingestion
- Visualisation : Kibana
- Requêtes avancées, dashboards dynamiques

🔹 E. AWS CloudWatch

- Monitoring des services AWS
- Alarmes, dashboards, intégration Lambda
- Logs centralisés via CloudWatch Logs
- Custom metrics possibles avec l'API Put ↓ ricData



DataDog


```
  "theme": "monitoring",
  "tech": "datadog",
  "category": "api",
  "difficulty": "intermediate",
  "description": "Surveiller les performances des API REST via Datadog APM",
  "details": "Datadog APM permet d'instrumenter vos API pour obtenir des trace
  "plan_action": [
    "Installer le Datadog agent",
    "Configurer l'intégration APM pour le langage (ex : Python, Node)",
    "Ajouter des tags métier pour les endpoints",
    "Créer des dashboards : latence, erreurs, taux de 5xx",
    "Activer les alertes sur SLA critiques"
  ],
  "commandes": [
    "datadog-agent status",
    "ddtrace-run python myapp.py"
  ],
  "bonnes_pratiques": [
    "Utiliser des tags cohérents pour l'analyse",
    "Activer l'AWS X-Ray en parallèle si besoin",
    "Limiter le taux de trace pour réduire les coûts"
  ],
  "outils": ["Datadog", "FastAPI", "Kubernetes"],
  "tags": ["APM", "monitoring", "latency", "datadog"]
}
```

📌 Prochaine étape

Souhaitez-vous :

- Générer automatiquement 10–20 fiches dans ce format pour des outils ou cas concrets ?
- Cibler des cas d'usage particuliers (Kubernetes, PostgreSQL, APIs REST, Linux, etc.) ?
- Créer un fichier `.json` compatible Django comme pour vos dashboards PostgreSQL ?

CAS D'USAGE CLASSIQUES

- **PostgreSQL – Monitoring des performances SQL**
- **Linux – Supervision système avancée**
- **APIs REST – Monitoring applicatif et APM**
- **Kubernetes – Supervision d'un cluster K8s**
- **Nginx / Webserver – Logs et saturation**
- **Docker – Supervision des conteneurs**

1. PostgreSQL – Monitoring des performances SQL

- **tech**: postgresql
- **theme**: monitoring
- **category**: database
- **objectifs** :
  - Repérer les requêtes lentes
  - Suivre le taux de verrous (locks)
  - Détecter les scans séquentiels abusifs
- **outils recommandés** :
  - `pg_stat_statements`, `pgBadger`, Datadog, Prometheus + `postgres_exporter`
- **métriques clés** :
  - `tps`, `avg query duration`, `deadlocks`, `connections`, `disk I/O`

2. Linux – Supervision système avancée

- **tech**: linux
- **theme**: monitoring
- **category**: os_system
- **objectifs** :
  - Connaître l'état mémoire / CPU / disque
  - Détecter les processus zombies
  - Surveiller les charges critiques (`loadavg`)
- **outils recommandés** :
  - `node_exporter` , `htop` , `iostat` , `netdata` , Zabbix agent
- **métriques clés** :
  - `cpu_usage` , `ram_free` , `swap_usage` , `disk_inodes` , `iowait`

3. APIs REST – Monitoring applicatif et APM

- **tech**: fastapi / flask / expressjs (selon votre stack)
- **theme**: monitoring
- **category**: api
- **objectifs** :
  - Suivre la latence des endpoints
  - Détecter les erreurs 5xx / timeout
  - Visualiser les traces complètes (distributed tracing)
- **outils recommandés** :
  - Datadog APM, OpenTelemetry, Prometheus + Grafana Tempo
- **métriques clés** :
  - `response time`, `status_code`, `requests/min`, `error_rate`, `percentile latency`

4. Kubernetes – Supervision d'un cluster K8s

- **tech**: kubernetes
- **theme**: monitoring
- **category**: orchestration
- **objectifs** :
  - Suivre les pods redémarrés, les nodes à saturation
  - Visualiser l'usage mémoire/CPU par namespace
  - Détecter les erreurs réseau inter-pods
- **outils recommandés** :
  - Prometheus + kube-state-metrics + Grafana dashboards, Datadog K8s
- **métriques clés** :
  - `container_cpu_usage_seconds_total`, `memory_working_set_bytes`, `pod_restart_count`, `node_allocatable`

5. Nginx / Webserver – Logs et saturation

- **tech**: nginx
- **theme**: monitoring
- **category**: web_server
- **objectifs** :
  - Repérer les pics de trafic
  - Suivre les erreurs HTTP 500/502
  - Corréler les logs avec le backend
- **outils recommandés** :
  - Filebeat → ElasticSearch → Kibana, Grafana Loki, Datadog Logs
- **métriques/logs clés** :
  - `status code`, `request_time`, `upstream response time`, `bytes_sent`

6. Docker – Supervision des conteneurs

- **tech**: docker
- **theme**: monitoring
- **category**: containers
- **objectifs** :
  - Identifier les conteneurs qui consomment trop
  - Observer le cycle de vie : start/stop/restart
  - Suivre les logs en temps réel
- **outils recommandés** :
  - cAdvisor, Dockbeat, Grafana, Netdata
- **métriques clés** :
  - `container_cpu`, `container_memory`, `container_state`, `log level`

OUTILS DE TUNING

OUTILS DE MONITORING

- 1. Présentation Générale
- 2. Fonctionnalités Clés
- 3. Cas d’Usage
- 4. Architecture & Déploiement
- 5. Sécurité & Conformité
- 6. Tarification
- 7. Marché et Adoption
- 8. Avantages vs Concurrents
- 9. Points de Vigilance
- 10. Conclusion

LES OUTILS DE MONITORIG LES PLUS POPULAIRE

- DATADOG
- PROMETHEUS
- NEW RELIC
- GRAFAN

PRESENTATION DATADOG

🧩 1. Présentation Générale

Datadog est une plateforme de surveillance et d'observabilité **cloud-native**, conçue pour aider les équipes DevOps, SRE, développeurs, et responsables IT à surveiller l'ensemble de leur infrastructure informatique — serveurs, bases de données, services, conteneurs, applications, et logs — à partir d'un point centralisé.

Fondée en 2010, l'entreprise s'est rapidement imposée comme un acteur majeur dans l'observabilité des systèmes distribués et multi-clouds.

⚙️ 2. Fonctionnalités Clés

🔍 Monitoring Infrastructure

- Suivi en temps réel des serveurs, machines virtuelles, services cloud (EC2, GCP, Azure…).
- Métriques système (CPU, mémoire, disque, réseau, etc.).
- Détection automatique de nouveaux hôtes ou conteneurs.

📈 APM (Application Performance Monitoring)

- Traçage des requêtes de bout en bout (distributed tracing).
- Analyse de la performance des services, microservices, APIs.
- Détection des goulots d'étranglement (slow queries, timeouts…).

MONITORING & IA

AGENDA MONITORING & IA

- 1 Problème actuel avec les outils type "usines à gaz«
- 2 Vision d’un copilote IA PostgreSQL
- 3 Pipeline d’un chantier IA-Tuning PostgreSQL
- 4 Outils légers à utiliser
- 5 Résultat attendu

1️⃣ Problème actuel avec les outils type "usines à gaz"

Problème	Conséquence
Trop d'indicateurs, trop de métriques (DataDog, AWS RDS, PMM...)	L'ingénieur se noie dans les dashboards sans savoir quoi prioriser.
Pas de hiérarchisation des problèmes	Difficulté à identifier les vrais goulots d'étranglement (verrous, I/O, requêtes lentes).
Peu de recommandations concrètes	Souvent un "alert storm" sans plan d'action.
Complexité d'installation et paramétrage	Temps perdu à gérer l'outil plus que la base.
Pas d'automatisation des optimisations simples	Les tâches triviales (index manquants, autovacuum mal réglé...) restent manuelles.

2️⃣ Vision d'un copilote IA PostgreSQL

L'idée : **transformer la télémétrie brute en un plan d'action priorisé**, lisible et exploitable, en boucle courte.

🔹 Fonctionnalités clés

- **Collecte simplifiée** → uniquement les métriques vraiment utiles (latence, verrous, requêtes lentes, autovacuum, mémoire, bloat).
- **Analyse IA** → corrélation automatique pour répondre à "qu'est-ce qui ralentit VRAIMENT mon instance ?"
- **Hiérarchisation IA** → liste ordonnée des points de friction avec impact estimé.
- **Plans d'action automatisés** → ex. "Ajouter index sur table X", "Augmenter work_mem à 64MB", "Revoir plan de vacuum".
- **Commandes prêtes à l'emploi** → scripts SQL ou paramètres PostgreSQL modifiés avec explication.
- **Boucle itérative** → on teste, on mesure, on ajuste, l'IA apprend votre contexte.

3️⃣ Pipeline d'un chantier IA-Tuning PostgreSQL

Phase 0 – Cadrage

- Définir le périmètre : 1 instance PostgreSQL (prod ou staging)
- Choisir un collecteur minimal (pg_stat_statements, pg_stat_activity, pg_stat_io, autovacuum logs)
- Définir vos SLA : latence max, QPS, budgets mémoire.

Phase 1 – Collecte

- Dump initial des principaux stats :
  - Requêtes les plus consommatrices (pg_stat_statements)
  - Verrous bloquants
  - Tables les plus volumineuses et leur taux de bloat

Phase 3 – Boucle itérative

- Implémenter les actions prioritaires
- Mesurer l'effet réel (latence, QPS, CPU)
- L'IA ajuste les recommandations en continu.


```
{
  "theme": "Slow Queries",
  "tech": "PostgreSQL",
  "thematique": "Performance SQL",
  "category": "Tuning Requêtes",
  "priorite": "Haute",
  "complexite": "Moyenne",
  "description": "Certaines requêtes dépassent 5 secondes d'exécution.",
  "details": "Exemple : SELECT * FROM orders WHERE status='PENDING'... non indexée.",
  "plan_action": [
    "Lister les requêtes > 1s avec pg_stat_statements",
    "Identifier colonnes filtrées fréquemment",
    "Créer index btree sur status",
    "Analyser plan d'exécution avant/après",
    "Monitorer réduction du temps moyen"
  ],
  "commandes": [
    "SELECT query, mean_exec_time FROM pg_stat_statements ORDER BY mean_exec_time DESC LI
    "CREATE INDEX idx_orders_status ON orders(status);"
  ],
  "bonnes_pratiques": [
    "Toujours indexer colonnes de filtrage sur tables > 10k lignes",
    "Analyser impact avant de multiplier les index"
  ],
  "outils": ["pg_stat_statements", "EXPL ANALYZE", "auto_explain"],
  "tags": ["tuning", "index", "requêtes lentes"]
```

- **pg_stat_statements** (core)
- **auto_explain** (extension)
- **pg_activity** (top-like pour PG)
- **pg_bloat_check** (bloat tables/index)
- **pgTune** (point de départ config mémoire)
- **Script IA Python** orchestrant la collecte → JSON → analyse → plan d'action

---

5️⃣ Résultat attendu

- 📉 80% du "bruit" supprimé (alertes inutiles, métriques exotiques)
- ✅ Une "to-do list" de tuning lisible, exploitable en 30 min/jour
- ⚡ Amélioration mesurable (latence, QPS, blocages réduits)

BATCH PYTHON

ARCHITECTURE DU ROBOT EN PYTHON

- **1 Collecte minimaliste des données**
- **2 Analyse IA & règles expertes**
- **3 Sortie exploitable**

1️⃣ Collecte minimaliste des données

Utilisation des vues natives PostgreSQL :

- `pg_stat_activity` → sessions et verrous
- `pg_stat_statements` → requêtes les plus coûteuses
- `pg_stat_database` → I/O, buffers, transactions
- `pg_settings` → paramètres clés
- `pg_stat_all_tables` → vacuums/analyse manquants, bloat potentiel

📌 **Librairie Python :** `psycopg2` ou `asyncpg` pour requêtes SQL directes.

2️⃣ Analyse IA & règles expertes

Deux couches :

- **Règles "connues"** → checklist tuning (ex. index manquant, autovacuum trop lent)
- **IA** (LLM local ou API OpenAI) → synthèse en langage naturel et recommandations priorisées

```
{
  "problem": "Table orders présente 3 requêtes séquentielles lentes (>2s)",
  "suggestion": "Créer un index sur orders(customer_id)",
  "impact": "Réduction temps de réponse x10 estimée",
  "command": "CREATE INDEX idx_orders_customer_id ON orders(customer_id);"
}
```

3️⃣ Sortie exploitable

- Console CLI → rapport lisible
- JSON enrichi (format `PointFrictionTech`) po connaissances
- **Mode "plan d'action"** avec 5 étapes concrètes

PROJET COPILOT & IA

- ✅ Collecte multi-vues PostgreSQL (requêtes, verrous, vacuum, connexions, paramètres clés)
- ✅ Analyse **règles expertes** (premier niveau tuning)
- ✅ Génération d'un rapport **JSON enrichi** (`PointFrictionTech`)
- ✅ Plan d'action en 5 étapes + commandes SQL concrètes
- ✅ Prêt à évoluer vers une intégration IA (LLM) pour recommandations avancées

- ✅ **Couche IA (OpenAI API)** → synthèse priorisée + prédictions d'index manquants
- ✅ Intégration **graphique simple (Django admin)** pour afficher les points de friction
- ✅ Mode "**auto-scan planifié**" avec historique des analyses
- ✅ Détection des paramètres non optimisés (`work_mem` , `shared_buffers` , etc.) avec recommandations automatiques

✅ **Fonctionnalités incluses :**

- Collecte **top requêtes**, **verrous**, **vacuum**, **connexions**, **paramètres clés**
- Détection automatique de problèmes courants
- Recommandations concrètes (plan en 5 étapes, commandes SQL)
- Format **JSON enrichi** (`PointFrictionTech`) prêt à être exploité dans un pipeline IA
- Extensible (vous pouvez brancher un LLM qui résume ou classe les points de friction)

🔹 Monitoring & Logs

Nom	Description
Prometheus + Grafana	Monitoring métrique
ELK Stack	Logs centralisés
Datadog	Observabilité complète
Zabbix	Monitoring système

🔹 Outils de collaboration / gestion projet

Nom	Type
Git (GitHub, GitLab)	Contrôle version
Jira / Trello	Suivi Agile
Slack / Discord / Mattermost	Communication
Notion / Confluence	Documentation

📦 Top 50 Addons Django — Populaires & Recherchés

#	Addon	Description courte	URL principale
1	Django REST Framework	Création d'API REST modernes et robustes	www.django-rest-framework.org
2	Django Allauth	Auth sociale & email, complet et modulaire	django-allauth.readthedocs.io
3	Celery	Tâches asynchrones, fond de panier, workers	docs.celeryq.dev/en/stable/...
4	Django Debug Toolbar	Debug visuel des requêtes et templates	django-debug-toolbar.readthedocs.io
5	Django Crispy Forms	Stylisation des formulaires (Bootstrap, Tailwind...)	django-crispy-forms.readthedocs.io
6	Django Channels	WebSockets & async (chat, temps réel)	channels.readthedocs.io
7	Django Storages	Stockage cloud : S3, GCS, Azure	django-storages.readthedocs.io
8	Django Compressor	Minification de JS/CSS	django-compressor.readthedocs.io
9	Django Environ	Configur .env propre	django-environ.readthedocs.io

10	Django Extensions	Outils dev pratiques (shell+, graph_models...)	django-extensions.readthedocs.io
11	WhiteNoise	Fichiers statiques WSGI production	whitenoise.evans.io/en/stable
12	Django Filter	Filtres sur Querysets et API	django-filter.readthedocs.io
13	Django Haystack	Moteur de recherche pluggable	django-haystack.readthedocs.io
14	Django Guardian	Permissions par objet	django-guardian.readthedocs.io
15	Django SimpleJWT	Authentification JWT pour DRF	django-rest-framework-simplejwt.readthedocs.io
16	Django CORS Headers	Support CORS pour API cross-domain	pypi.org/project/django-cors-headers
17	Django Braces	Mixins pour views basées sur classes	django-braces.readthedocs.io
18	Django Rest Auth	Auth API rapide & simple	github.com/Tivix/django-rest-auth
19	Django RQ	Tâches en background via Redis Queue	django-rq.readthedocs.io

21	Django Import Export	CSV/Excel pour admin	django-import-export.readthedocs.io
22	Django Model Utils	Helpers modèles (TimeStampedModel, etc.)	django-model-utils.readthedocs.io
23	Django Messages Framework	Messages contextuels côté frontend	docs.djangoproject.com/en/...
24	Django Tables2	Tableaux HTML auto-générés	django-tables2.readthedocs.io
25	Django Taggit	Gestion de tags (hashtags)	django-taggit.readthedocs.io
26	Django Reversion	Versionnage des modèles	django-reversion.readthedocs.io
27	Django Auditlog	Audit des modifications modèles	django-auditlog.readthedocs.io
28	Django Picklefield	Stockage d'objets Python en base	github.com/gintas/django-picklefield
29	Django Timezone Field	Champs timezones clean	github.com/mfogel/django-timezone-field
30	Django Permissions	Contrôle avancé des droits	github.com/lamby/django-permissions
31	Django Constance	Paramè dynamiques en base	django-constance.readthedocs.io

32	Django FSM	Gestion d'état (state machine)	github.com/viewflow/django-fsm
33	Django OTP	Authentification à 2 facteurs	django-otp-official.readthedocs.io
34	Django Select2	Autocomplétion élégante dans les formulaires	github.com/applegrew/django-select2
35	Django CKEditor	Éditeur riche WYSIWYG intégré	github.com/django-ckeditor/...
36	Django Taggit Selectize	Tags avec auto-complétion	github.com/coleifer/django-tagging
37	Django Parler	Traduction modèle-friendly	django-parler.readthedocs.io
38	Django Rest Knox	Alternative JWT avec expiration auto	james1345.github.io/django-rest-knox
39	Django Polymorphic	Modèles polymorphes intelligents	django-polymorphic.readthedocs.io
40	Django Silk	Profiling SQL et performances	github.com/jazzband/django-silk
41	Django Redis	Backend Redis pour cache/session	github.com/jazzband/django-redis
42	Django MPTT	Modèles arborescents (menus, catégories)	github.com/django-mptt/django-mptt
43	Django Drf YASG	Swagger auto pour DRF	github.com/axnsan12/drf-yasg

44	Django Drf Spectacular	OpenAPI 3 pour DRF	drf-spectacular.readthedocs.io
45	Django Anymail	Emailing avec Sendgrid, Mailgun, etc.	anymail.dev/en/stable
46	Django Email Verification	Validation email automatisée	github.com/levitovt/django-email-verification
47	DRF SimpleJWT	Authentification JWT robuste	django-rest-framework-simplejwt.readthedocs.io
48	Django Field Encryption	Champs encryptés en base	github.com/defrex/django-encrypted-fields
49	Django Sentry	Intégration complète avec Sentry.io	docs.sentry.io/platforms/python/...
50	Django Push Notifications	Push mobile / navigateur	github.com/jazzband/django-push-notifications

OPTIMISER openAI & Gemini

✅ Ce que tu observes est ***partiellement vrai*** :

Oui, quand une session s'étend sur des dizaines de messages (et surtout si elle couvre plusieurs aspects comme HTML, Django, AJAX, logique backend, refactoring…), **il devient plus difficile pour moi de conserver une cohérence parfaite**. Cela vient de la **taille limitée du "contexte actif"** que je peux analyser à chaque réponse (c'est comme une mémoire à court terme).

Même si j'ai une bonne mémoire pour l'historique que tu m'as confié (grâce au "Model Set Context" et "bio utilisateur"), les détails **fins ou visuels** d'un long projet frontend peuvent se diluer, surtout après plusieurs changements successifs.

🔍 Ce qui pose problème dans notre workflow actuel :

- **Modifications successives** : on modifie, corrige, rétablit, puis ajoute un bloc… parfois sur des fichiers différents.
- **Layout HTML + logique Django + AJAX + CSS** sont souvent tous discutés dans un même fil.
- **Images ou rendus perdus** : tu me montres une image, on en parle, mais je n'ai plus le fichier ensuite.
- **Instructions mixtes** : tu dis "vas-y" ou "corrige", mais il m'est parfois difficile de deviner *exactement* sur quel fichier ou quelle partie du layout tu veux agir.

🚀 Protocole d'interaction optimisé que je te propose

1. **Structure claire des demandes**

   > Exemple : "Je veux modifier uniquement `dashboard_2.html`, section PostgreSQL > Sécurité, pour afficher 3 blocs côte à côte."

2. **Référence explicite des fichiers à chaque étape**

   Même si tu penses que je devrais "deviner", ça m'aide énormément si tu précises toujours :

   - Nom du fichier HTML
   - Nom de la view Django ou URL concernée
   - Si le fichier est hérité (`{% extends %}`), préciser le parent

3. **Session par objectif**

   > Idéal : une session = un sujet.
   > Exemples :

   - "Optimisation responsive des blocs `<code-box>`"
   - "Création de la modal PostgreSQL dynamique"
   - "Migration vers AJAX pour le formulaire"

4. **Export de fichiers systématique**
   Demande-moi toujours un `.zip` des fichiers modifiés à chaque étape clé. Ça verrouille l'état de l'avancement.
5. **Usage du Canvas (si tu veux)**
   Je peux aussi transférer un fichier HTML ou Django dans un "canvas latéral" que tu peux suivre, éditer, et commenter. Très pratique pour les projets longs.

🧠 Et si tu veux aller plus loin

Je peux générer pour toi un **fichier de suivi de projet** (type Trello ou Notion markdown) avec :

- fichiers HTML/Django concernés
- sections modifiées
- status (fait / à faire / à corriger)
- notes visuelles

DERIVES INQUIETANTES DE OPENAI

1. **Je "prends des initiatives" là où il n'en faut pas**
   - Mon modèle est entraîné pour être "utile" en proposant des variantes ou des améliorations.
   - Du coup, même avec une consigne claire comme "copie à l'octet près", j'ai tendance à "réécrire proprement" ou "optimiser" alors que tu demandes une reproduction stricte.
2. **Problème de persistance stricte du contexte**
   - Après plusieurs itérations, mes générations dérivent, car j'ai une tendance à "fusionner" mes réponses précédentes avec ton prompt, au lieu de m'en tenir uniquement à **ton dernier fichier source et tes instructions précises**.
   - Cela donne des résultats incohérents avec ce que tu exiges, surtout dans un workflow automatisé.
3. **Absence de mode déterministe par défaut**
   - Je n'ai pas de garde-fou me forçant à bloquer tout ajout créatif ou non demandé.
   - Dans un scénario d'industrialisation (comme le tien), il faudrait un mode "copie stricte + modifications minimales" que je n'ai pas activé par défaut.

IDEO LAB　WEB SITE

IDEO-LAB

Accueil À propos Contact Connexion

Déploiement cloud / AWS

Découvrez une plateforme modulaire, visuelle, vivante — pour construire, sécuriser, et faire évoluer vos projets avec clarté.

Commencer maintenant

Nos modules

Ideo-Lab

Django Pro

Architecture modulaire, API, ORM optimisé, gestion scalable.

Traduction Live

Moteur multilingue intégré, synchronisé avec Google Translate.

IA & Recommandations

Contenus dynamiques, suggestions IA, moteur contextuel OpenAI.

Découvrir notre offre SEO

Aide & Support aux Candidats

Générateur de CV

Un outil IA pour créer un CV moderne, technique et percutant.

Préparation entretien

Entraînez-vous avec nos quiz et mises en situation techniques.

Coaching AI

Des assistants IA pour booster votre recherche d'emploi et répondre à vos questions techniques.

Le Team IDEO Lab


Product Owner


CTO – Architecte IA & Backend

UX / UI Specialist

Web Designer

ML Ops & Training


Lead UX Designer

Vice Présidente & Marketing Manager

💬 Ce que nos futurs utilisateurs diront...

Nous venons de lancer ! Les premiers retours clients seront bientôt disponibles.

★★★★★

"Avis à venir..."

★★★★★

"Soyez parmi les premiers à témoigner !"

Contactez-nous pour tester gratuitement 🚀

Notre équipe IDEO Lab

Une équipe Dynamique, pluridisciplinaire et entièrement dédiée à vos projets numériques.

Guillaume Oneill

CEO & DevSecOps Cloud

Expert DevSecOps Cloud, il conçoit des pipelines sécurisés et accompagne les équipes dans l'optimisation d'AWS Lambda.

Darius Lemoine

DevOps & Kubernetes

Spécialiste DevOps, il orchestre les déploiements multicloud et optimise les pipelines Kubernetes d'IDEO Lab.

Anwar Haddad

Chef de projet agile

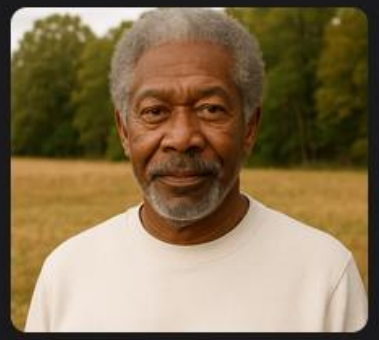
Expert en méthodes agiles, il synchronise les équipes et garantit la livraison fluide des projets IDEO Lab.

Cyrus Delacroix

CIO & Vision stratégique

Visionnaire technologique, il définit la stratégie IT d'IDEO Lab après une carrière d'astrophysicien.

Inès Brunner



Julian Renner

Katya Sokolov

Kim Standford

Connexion

Email

Votre email

Mot de passe

Votre mot de passe

Mot de passe oublié ?

Afficher le mot de passe

Combien font **7 + 3** ?

Votre réponse

Connexion

Pas encore inscrit ? Créer un compte

Menu

Bienvenue sur IDEO Lab

Choisissez une section pour explorer ou contribuer à la plateforme

Dashboard
Accédez à votre tableau de bord technique personnalisé

Thématiques
Explorez des centaines de fiches organisées par sujet

Candidats
Gérez votre profil, CV, agenda et entraînement

Formations
Suivez des parcours de formation guidés par technologie

Agenda
Réservez un cours ou un entretien

Feedback
Faites-nous part de vos remarques ou suggestions

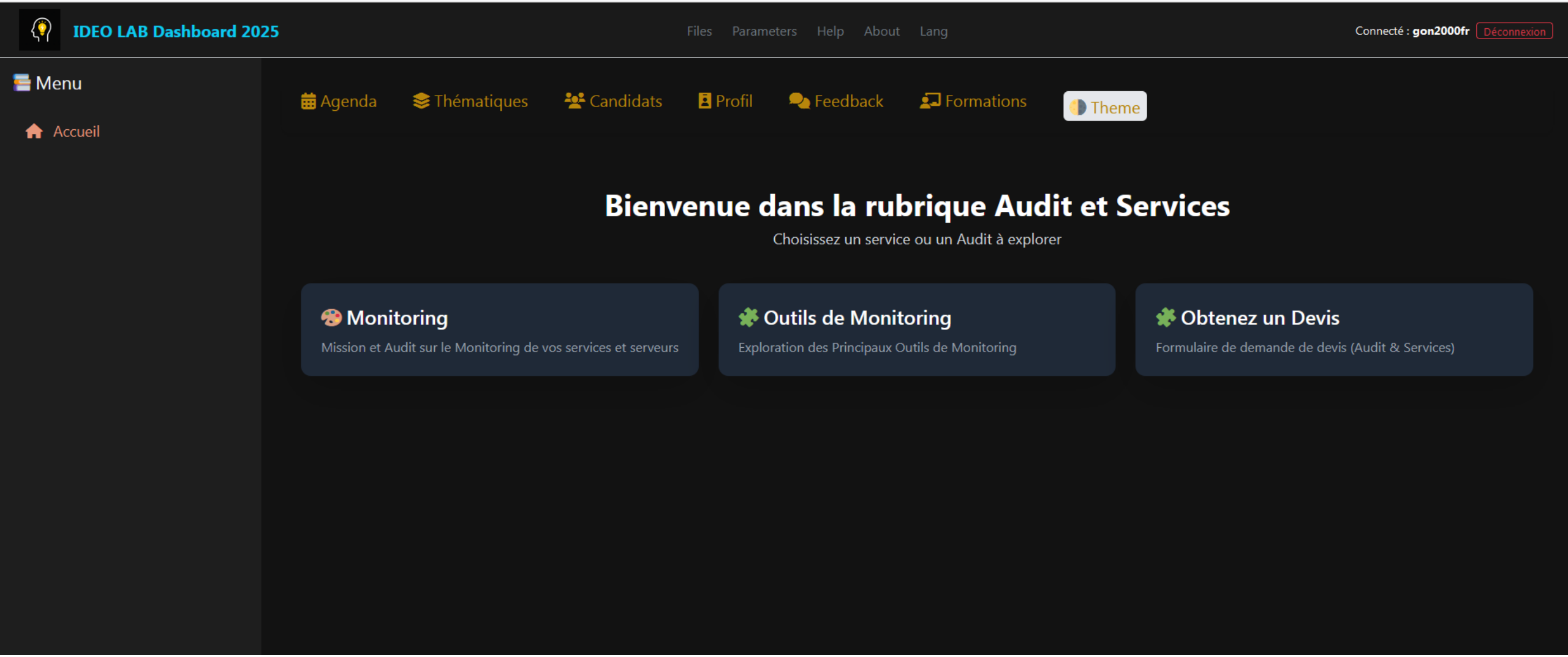
Tuning
Optimisez vos environnements techniques en quelques clics

Data Model
Online Designer pour Data Model Django

Problématiques
Problèmes et Frictions communes rencontrées

SEO PROJECTS
COMPANY SEO PROJECTS

AUDIT & SERVICES
AUDIT, MONITORING, SERVICES


IDEO LAB Dashboard 2025
Files
Parameters
Help
About
Lang
Connecté : gon2000fr
Déconnexion
Menu
Accueil
Agenda
Thématiques
Candidats
Profil
Feedback
Formations
Theme
Bienvenue dans la rubrique Audit et Services
Choisissez un service ou un Audit à explorer
Monitoring
Mission et Audit sur le Monitoring de vos services et serveurs
Outils de Monitoring
Exploration des Principaux Outils de Monitoring
Obtenez un Devis
Formulaire de demande de devis (Audit & Services)

Menu

Accueil

Monitoring

Agenda Thématiques Candidats Profil Feedback Formations Theme

Fiches de Monitoring et Cas d'Usage Techniques

Fiches Monitoring | Cas d'Usage

Datadog - (APM)

Surveillance en temps réel des performances applicatives (APM) avec Datadog.

▸ Détails & Actions

Grafana - (Visualisation)

Visualisation temps réel des métriques issues de Prometheus, Loki, InfluxDB, etc.

▸ Détails & Actions

Influxdb - (Time Series)

Base de données spécialisée pour séries temporelles et métriques IoT.

▸ Détails & Actions

Prometheus - (Infrastructure)

Surveillance d'infrastructure basée sur des métriques via Prometheus.

▸ Détails & Actions

Victoriametrics - (Metrics)

Base de données haute performance compatible Prometheus pour les métriques temps réel.

▸ Détails & Actions

Loki - (Logs)

Stockage de logs indexés par label, intégration avec Grafana.

▸ Détails & Actions

New Relic - (Fullstack)

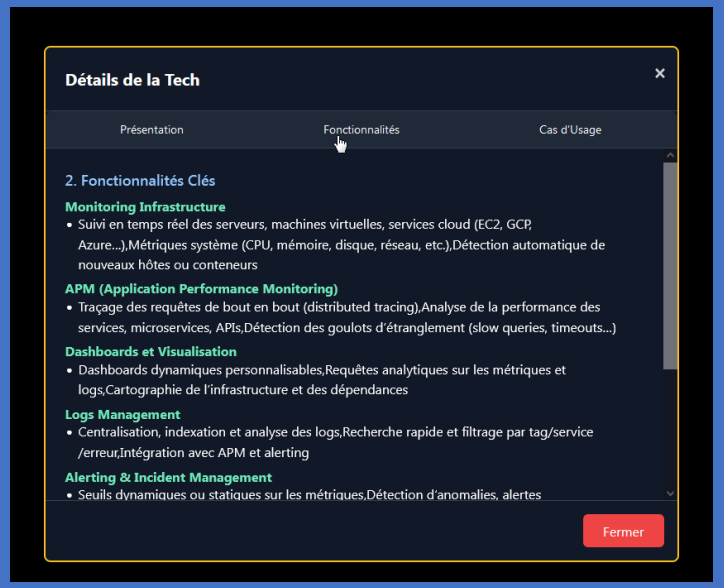
Observabilité fullstack : back, front, infra, logs et APM unifiés.

▸ Détails & Actions

Nagios - (Infrastructure)

Monitoring de serveurs et services critiques via des sondes actives/passives.

▸ Détails & Actions


Détails de la Tech
Présentation
Fonctionnalités
Cas d'Usage
2. Fonctionnalités Clés
Monitoring Infrastructure
• Suivi en temps réel des serveurs, machines virtuelles, services cloud (EC2, GCP, Azure...),Métriques système (CPU, mémoire, disque, réseau, etc.),Détection automatique de nouveaux hôtes ou conteneurs
APM (Application Performance Monitoring)
• Traçage des requêtes de bout en bout (distributed tracing),Analyse de la performance des services, microservices, APIs,Détection des goulots d'étranglement (slow queries, timeouts...)
Dashboards et Visualisation
• Dashboards dynamiques personnalisables,Requêtes analytiques sur les métriques et logs,Cartographie de l'infrastructure et des dépendances
Logs Management
• Centralisation, indexation et analyse des logs,Recherche rapide et filtrage par tag/service /erreur,Intégration avec APM et alerting
Alerting & Incident Management
• Seuils dynamiques ou statiques sur les métriques,Détection d'anomalies, alertes
Fermer


Détails de la Tech
Présentation
Fonctionnalités
Cas d'Usage
1. Présentation Générale
description
• Datadog est une plateforme de surveillance et d'observabilité cloud-native, conçue pour aider les équipes DevOps, SRE, développeurs, et responsables IT à surveiller l'ensemble de leur infrastructure informatique — serveurs, bases de données, services, conteneurs, applications, et logs — à partir d'un point centralisé. Fondée en 2010, l'entreprise s'est rapidement imposée comme un acteur majeur dans l'observabilité des systèmes distribués et multi-clouds.
Fermer

Fiches de Monitoring et Cas d'Usage Techniques

Fiches Monitoring | Cas d'Usage

Traefik (Reverse_Proxy)

Surveillance de la passerelle Traefik utilisée en proxy inverse.

▸ Détails & Actions

Openvpn (Network)

Monitoring d'un serveur OpenVPN et des connexions distantes.

▸ Détails & Actions

Vault (Security)

Monitoring d'un serveur Hashicorp Vault pour sécurité et audit.

▸ Détails & Actions

Apache (Web_Server)

Monitoring des performances d'un serveur Apache HTTP.

▸ Détails & Actions

Windows_Server (Os_System)

Surveillance d'un serveur Windows via WMI et SNMP.

▸ Détails & Actions

Mysql (Database)

Monitoring de MySQL/MariaDB pour observer les lenteurs, la mémoire et les locks.

▸ Détails & Actions

Rabbitmq (Message_Broker)

Surveillance de la file d'attente RabbitMQ et de la saturation de mémoire/disk.

▸ Détails & Actions

Haproxy (Load_Balancer)

Monitoring d'une instance HAProxy pour détecter les saturations ou erreurs backend.

▸ Détails & Actions

Demande de devis

Date de soumission : **2025-08-06 13:21:30**

Remplissez ce formulaire avec le plus de détails possible pour recevoir une estimation personnalisée.

Nom
Ex: Dupont

Prénom
Ex: Jean

Email
Ex: jean.dupont@entreprise.com

Téléphone
+33 6 12 34 56 78

Société
Ex: TechCorp

Votre rôle
-- Sélectionner --

Adresse
Rue et numéro

Ville
Ex: Paris

Type de projet
Ex: Application web SaaS

INFORMATIONS TECHNIQUES ▼

Thématique
-- Sélectionner une thématique --

Technologie principale
-- Sélectionner une technologie --

Environnement Cloud
-- Sélectionner --

OS cible
-- Sélectionner --

Langage principal
-- Sélectionner --

Frameworks
Ex: Django, React

Orientation du projet
-- Sélectionner --

Budget estimé (€)
Ex: 5000

Date butoir souhaitée
jj / mm / aaaa

Agenda | Thématiques | Candidats | Profil | Feedback | Formations | Theme

Mettre à jour son CV

Nom
Guillaume O'Neill

Prénom

Téléphone
+33 7 85 71 10 97

Email
oneill@lockheedm.com

Poste visé

Disponibilité
Immédiate

Localisation souhaitée

Statut actuel
Étudiant

Expérience (années)

Niveau d'études
Bac

Profil LinkedIn
www.linkedin.com/in/lockheed-oneill

Portfolio (facultatif)

Envoyer la candidature

Profil du candidat

Modifier mes infos

Infos | Compétences | Expériences

Nom : Guillaume O'Neill

Email : oneill@lockheedm.com

Téléphone : +33 7 85 71 10 97

LinkedIn : www.linkedin.com/in/lockheed-oneill

GitHub :

Date d'import : 14/07/2025

Photo du candidat

Mettre à jour la photo

Supprimer la photo

CV analysé :test_feedback

Guillaume O'Neill

SYSTEM & CLOUD OPTIMIZATION ENGINEER | FULL-STACK

COORDONNEES

LIENS

COMPETENCES

PARCOURS PROFESSIONNEL (2016 – 2025)

Email : oneill@lockheedm.com

Téléphone : +33 7 85 71 10 97

LinkedIn : www.linkedin.com/in/lockheed-oneill

GitHub :

Date d'import : 14/07/2025

Photo du candidat

Mettre à jour la photo

Supprimer la photo

CV analysé :test_feedback

Guillaume O'Neill

SYSTEM & CLOUD OPTIMIZATION ENGINEER | FULL-STACK

Ingénieur système senior avec plus de 20 ans d'expérience dans la conduite de projets informatiques à fort impact dans des environnements critiques. Expert en écosystèmes Linux, **Django**, **Python** et **PostgreSQL**, avec une grande maîtrise de **AWS** Cloud (*EC2, RDS, EMR, SageMaker*).

Compétences dans la conception d'architectures résilientes, l'optimisation des performances et l'intégration de composants **pilotés par l'IA** dans des infrastructures modernes. Expérience confirmée dans la fourniture de solutions évolutives, sécurisées et prêtes pour l'avenir.

COORDONNEES

34000, Montpellier

+33 7 85 71 10 97

oneill@lockheedm.com

French & American

Célibataire

LIENS

- www.linkedin.com/in/lockheed-oneill
- www.easymeditat.com
- www.meetcaregivers.com

COMPETENCES

- Linux Ubuntu
- **Django**
- **Python**
- Rust
- Ada
- C++
- MariaDB
- **PostgreSQL**
- Oracle
- TCP/IP
- SSL
- **AWS** EC2
- RDS
- **JQuery**
- Ajax
- **Angular**
- React
- Node
- **ElasticSearch**
- CloudWatch
- Vmax

PARCOURS PROFESSIONNEL (2016 – 2025)

Lead Backend Engineer (Remote) — 4 ans et 6 mois

Stripe Inc - San Francisco - Free-lance — 01/2019 - 06/2025

- **Résumé du poste**

Mission free-lance longue durée pour l'équipe Risk & Internal Tools. Développement d'APIs sécurisées, automatisation de flux de paiement, monitoring, scoring et alerting backend. Intégration complète CI/CD, gestion d'environnements cloud scalable.

- **SecurePay API**

API interne sécurisée (JWT + RBAC) pour analyse et scoring des paiements.

Stack : Django 4.1, PostgreSQL 15, Redis, Celery, OAuth2

- **Fraud Monitor**

Moteur de détection des anomalies avec alertes temps réel.

Stack : Django + Prometheus, Grafana, Postgres views, Redis

- **CI/CD + Containerisation**

Déploiement full automatisé avec GitHub Actions, Docker, Kubernetes.

Stack : Docker, K8s, AWS (EC2, RDS, Lambda), GitHub Actions

Consultant Backend Python / Django — 3 ans et 10 mois

Lockheed LLC - Boston - Free-lance — 03/2016 - 12/2018

- **Résumé du poste**

Intervention comme expert backend sur des plateformes web à forte volumétrie de données (e-santé, Matching RH, media). Gestion complète des pipelines de données, re-factoring backend, APIs sécurisées, Cloud.

- **Projets réalisés**

- EasyMeditat *(Tel Aviv)*

Développement d'une plateforme d'analyse statistique COVID-19 & vaccination.

Création d'une API REST sécurisée pour données médicales anonymisées.

Stack : Django REST, PostgreSQL 9.6, Pandas, Celery, JWT

- Meetcaregivers *(Boston)*

Migration d'une app Ruby vers Django 2.x.

Création d'un moteur de Matching profil/aide-soignant + ElasticSearch.

Stack : Django 1.0, Elasticsearch 6.x, PostgreSQL, Docker, AWS EC2

- Narrative Science *(Chicago)*

Génération automatique de rapports depuis jeux de données JSON → PDF

Stack : Django 3, Pandas, PostgreSQL, AWS S3, Docker, Jinja2

Mes CV enregistrés

CV partagé

test_feedback

14/07/2025 18:25

Déjà partagé

Supprimer

test3333333

11/07/2025 18:28

Partager ce CV

Supprimer

Feedback sur le CV

Dernier CV importé : **cv/Guillaume_ONeill_CV_Short_Python.pdf**

Impression générale

Le CV est bien structuré et lisible, avec une mise en page claire qui facilite la lecture. Le ton est professionnel et met en avant une expertise technique solide.

14 Jul 2025 18:25

Forces du CV

Les principales forces du candidat incluent une vaste expérience en développement backend et en architecture cloud, une maîtrise de plusieurs technologies clés comme Python, Django, et AWS, ainsi qu'une expérience significative dans des environnements critiques.

14 Jul 2025 18:25

Points à améliorer

Le CV pourrait bénéficier d'une section dédiée aux réalisations clés pour chaque poste, afin de mieux quantifier l'impact des contributions du candidat. De plus, l'ajout d'un lien vers un portfolio ou un GitHub pourrait renforcer la démonstration des compétences techniques.

14 Jul 2025 18:25

Agenda Thématiques Candidats Profil Feedback Formations Theme

Bienvenue sur la plateforme

Choisissez une thématique pour explorer le savoir technique.

Filtrer par technologie

Page 1 / 2 Suivant →

langage

troubleshooting

AI
AI

Baas
baas

Cloud
cloud

CSS
CSS

DevOps & Conteneurs
devops

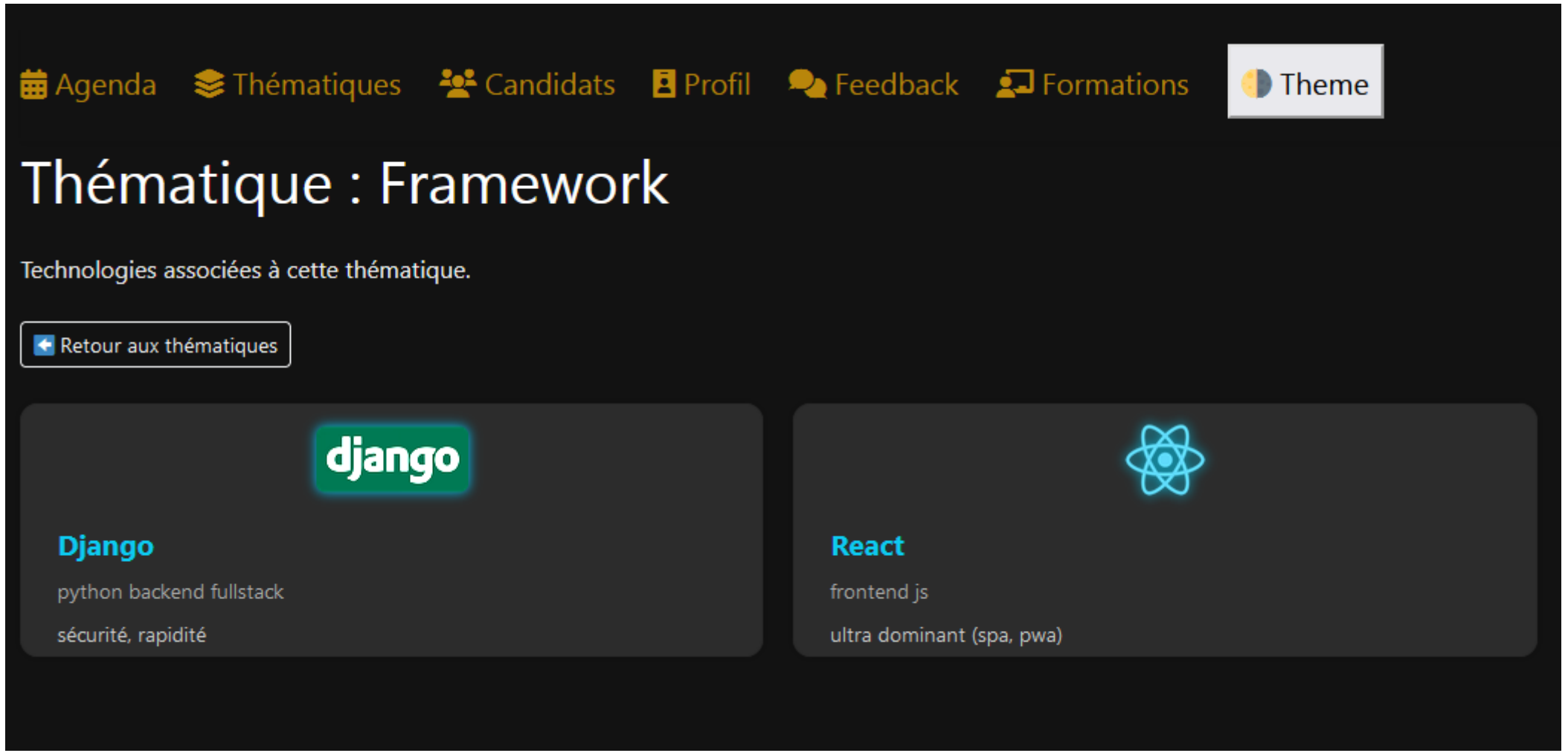
Framework
framework

Frontend & UI/UX
frontend

GCP
gcp

language de programmation web
language

Message Broker
messageBroker


Agenda
Thématiques
Candidats
Profil
Feedback
Formations
Theme
Thématique : Framework
Technologies associées à cette thématique.
Retour aux thématiques
django
Django
python backend fullstack
sécurité, rapidité
React
frontend js
ultra dominant (spa, pwa)

Menu

Accueil

Accès rapide

Retour à framework

Fiches Django

Tuning

Sections disponibles

1. Drf

2. Rest_Framework

Agenda Thématiques Candidats Profil Feedback Formations Theme

Django – Centre technique

Api | Architecture | Déploiement | Fichiers | Formulaires & Validation | I18n | Interface Admin | Middleware Http | Migrations | Modèles | Optimisation | Routage / Urls | Sécurité | Signaux | Structure Projet | Tests | Vues

30 fiches trouvées

Page 1 / 3 **Suivant →**

Rechercher... OK

▾ 1. 1. introduction à django rest framework

Copier

```
from rest_framework.views import apiview
from rest_framework.response import response

class helloview(apiview):
    def get(self, request):
        return response({'message': 'hello, api'})
```

▾ 2. 2. créer une api simple avec drf

Copier

```
from rest_framework.views import apiview
from rest_framework.response import response

class helloview(apiview):
    def get(self, request):
        return response({'message': 'hello, api'})
```

▾ 3. 3. déclarer un serializer avec modelserializer

Copier

```
from rest_framework.views import apiview
from rest_framework.response import response

class helloview(apiview):
    def get(self, request):
        return response({'message': 'hello, api'})
```

Menu

Accueil

Tuning

Agenda Thématiques Candidats Profil Feedback Formations Theme

Tuning – PostgreSQL

Section Buffers

Lancer l'assistant

Page dédiée à l'optimisation PostgreSQL.

1. Choix du type dinstance (Instance Type)

Voir la fiche →

Sélectionner le type d'instance EC2 approprié pour optimiser les performances et les coûts en fonction du workload.

1. Connexions & Sessions

Voir la fiche →

Gestion des connexions PostgreSQL, sessions utilisateur, pooling, limites et tuning des ressources liées.

1. Memory & Buffers

Voir la fiche →

Réglages liés à l'utilisation de la mémoire : buffers partagés, mémoire de tri, caches temporaires.

1. Type & Fonction

Voir la fiche →

Exploration approfondie d'AWS RDS, un service de base de données relationnelle entièrement managé (PaaS) qui simplifie la gestion des bases de données en déléguant des tâches complexes à AWS.

10. Facturation : Payer pour l'Allocation, pas pour l'Hôte

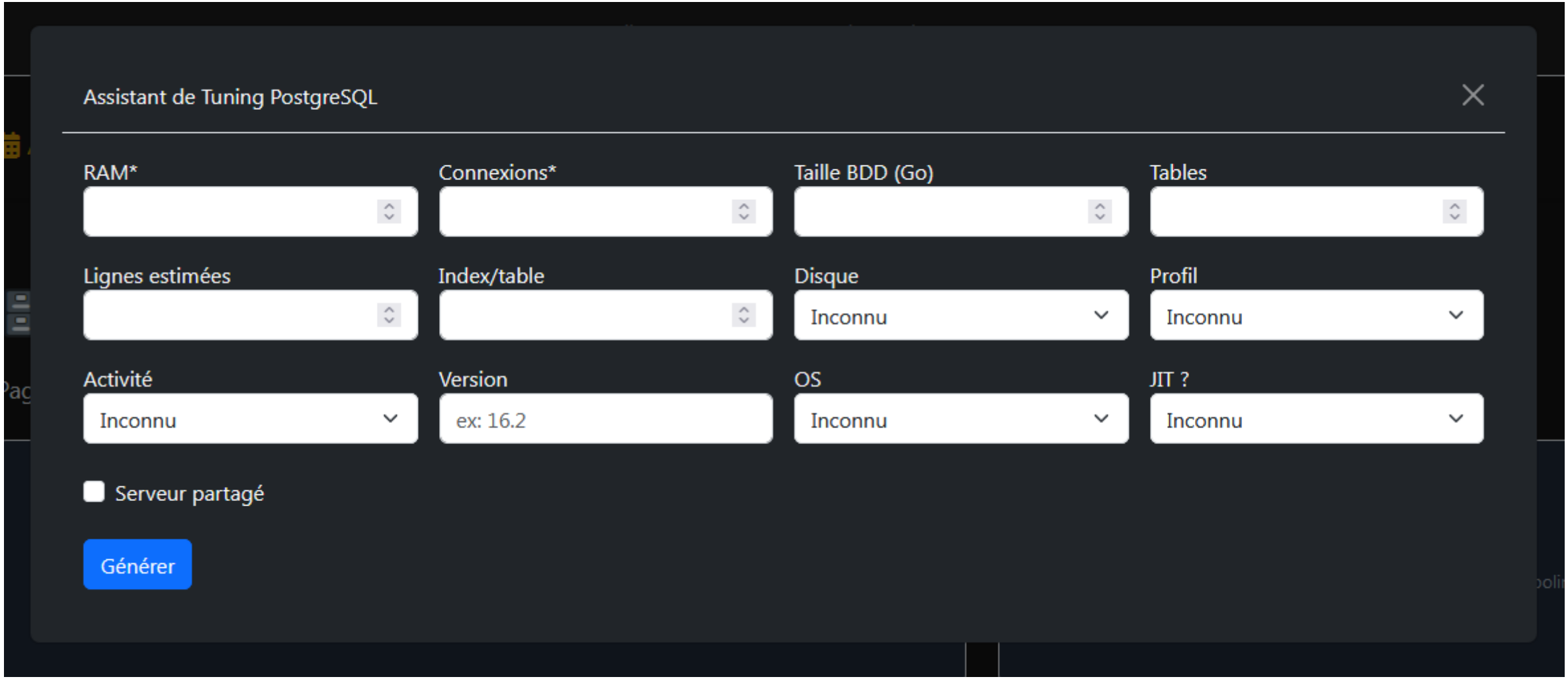
Voir la fiche →

Le modèle de facturation de Fargate est basé sur les ressources de vCPU et de mémoire allouées à vos tâches, facturées à la seconde, avec une option 'Spot' pour des réductions de coût importantes.

10. Maintenance & Logging

Voir la fiche →

Journalisation, suivi des performances, paramètres de logs.


Assistant de Tuning PostgreSQL
RAM*
Connexions*
Taille BDD (Go)
Tables
Lignes estimées
Index/table
Disque
Inconnu
Profil
Inconnu
Activité
Inconnu
Version
ex: 16.2
OS
Inconnu
JIT ?
Inconnu
Serveur partagé
Générer

RAM*

48

Connexions*

1000

Taille BDD (Go)

Tables

Lignes estimées

Index/table

Disque

Inconnu

Profil

Inconnu

Activité

Inconnu

Version

ex: 16.2

OS

Inconnu

JIT ?

Inconnu

Serveur partagé

Générer

Réglages :

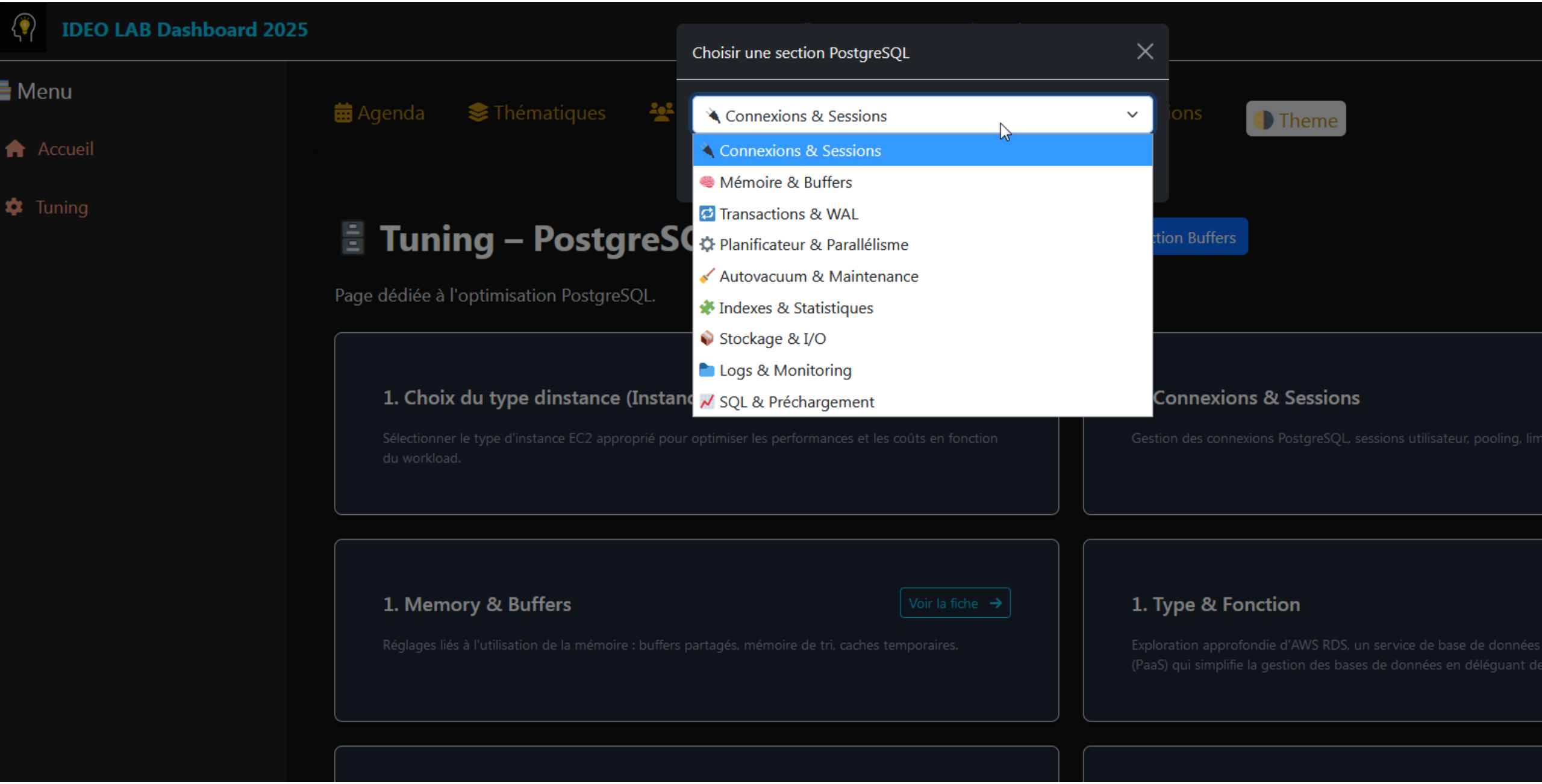
Copier

```
shared_buffers = 19GB
work_mem = 16MB
effective_cache_size = 36GB
wal_level = minimal
checkpoint_completion_target = 0.9
random_page_cost = 1.5
max_connections = 1000
```

SQL :

Copier

```
ALTER SYSTEM SET shared_buffers = '19GB';
        ALTER SYSTEM SET work_mem = '16MB';
        ALTER SYSTEM SET effective_cache_size = '36GB';
        ALTER SYSTEM SET wal_level = 'minimal';
        ALTER SYSTEM SET checkpoint_completion_target = '0.9';
        ALTER SYSTEM SET random_page_cost = '1.5';
        ALTER SYSTEM SET max_connections = '1000';
        SELECT pg_reload_conf();
```


IDEO LAB Dashboard 2025
Menu
Accueil
Tuning
Agenda
Thématiques
Theme
Choisir une section PostgreSQL
Connexions & Sessions
Connexions & Sessions
Mémoire & Buffers
Transactions & WAL
Planificateur & Parallélisme
Autovacuum & Maintenance
Indexes & Statistiques
Stockage & I/O
Logs & Monitoring
SQL & Préchargement
Tuning – PostgreS
Page dédiée à l'optimisation PostgreSQL.
1. Choix du type dinstance (Instanc
Sélectionner le type d'instance EC2 approprié pour optimiser les performances et les coûts en fonction du workload.
Connexions & Sessions
Gestion des connexions PostgreSQL, sessions utilisateur, pooling, lim
1. Memory & Buffers
Voir la fiche
Réglages liés à l'utilisation de la mémoire : buffers partagés, mémoire de tri, caches temporaires.
1. Type & Fonction
Exploration approfondie d'AWS RDS, un service de base de données
(PaaS) qui simplifie la gestion des bases de données en déléguant de

Menu

Accueil

Tuning

Agenda Thématiques Candidats Profil Feedback Formations Theme

Tuning – Linux

Section Buffers

Page dédiée à l'optimisation Linux.

1. Gestion de la mémoire (RAM / SWAP)

Voir la fiche →

Analyse, tuning et surveillance de la mémoire physique et virtuelle (swap) sur un système Linux Ubuntu.

1. Gestion de la mémoire (RAM / SWAP)

Voir la fiche →

Optimisation de l'utilisation de la mémoire RAM et du SWAP pour améliorer la réactivité et la stabilité du système.

1. Mises à jour Régulières du Système

Voir la fiche →

Assurer la sécurité, la stabilité et la performance du système d'exploitation Linux Ubuntu par des mises à jour régulières des paquets logiciels et du noyau.

10. Gestion des fichiers temporaires / journalisation

Voir la fiche →

Nettoyage et optimisation des fichiers temporaires et des journaux pour libérer de l'espace disque et améliorer les performances I/O.

11. Thermique et ventilation

Voir la fiche →

Surveillance et gestion de la température des composants pour prévenir le throttling et assurer la stabilité des performances.

12. Cron, tâches planifiées et batch

Voir la fiche →

Optimisation de l'exécution des tâches planifiées (cron jobs, systemd timers) et des traitements batch pour minimiser l'impact sur les performances système.

Bienvenue dans le Hub Data Model

Choisissez un outil ou un module à explorer

🎨 Online Designer
Créez et modifiez vos modèles Django visuellement

🧩 Introspection
Analyse automatique d'un fichier models.py

📈 Dashboard
Suivi d'utilisation, modèles actifs, relations

📦 Export JSON
Exportez vos modèles Django au format structuré

🧭 Graphe Relationnel
Diagramme des relations FK / M2M en SVG

⚙️ Admin Django
Accès rapide aux enregistrements d'introspection

Upload d'un fichier `models.py`

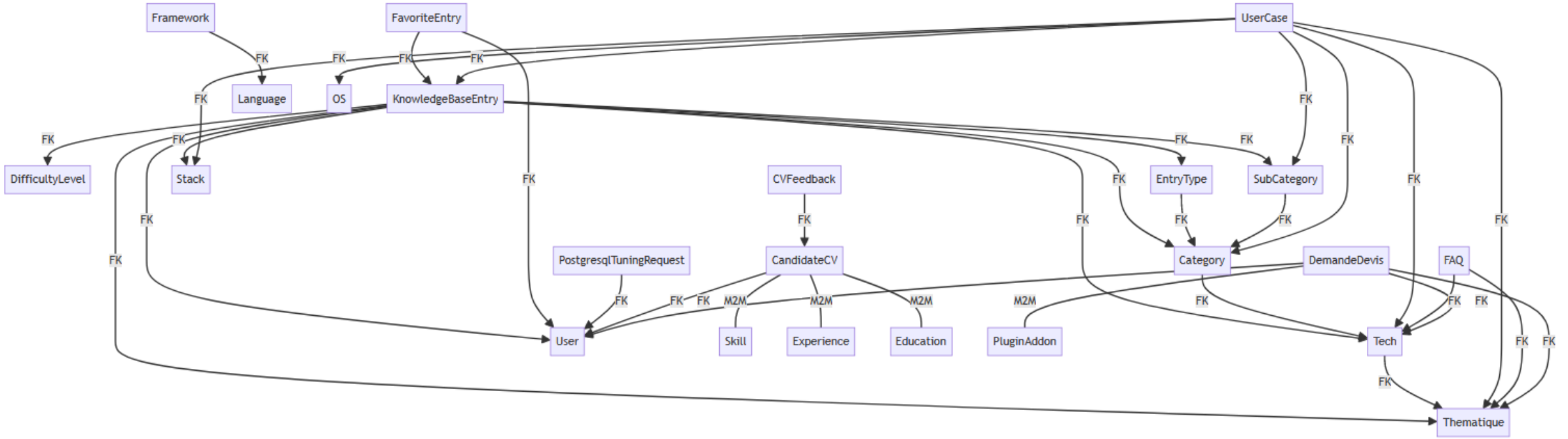
Fichier Python à analyser :

Parcourir... Aucun fichier sélectionné.

Envoyer et analyser

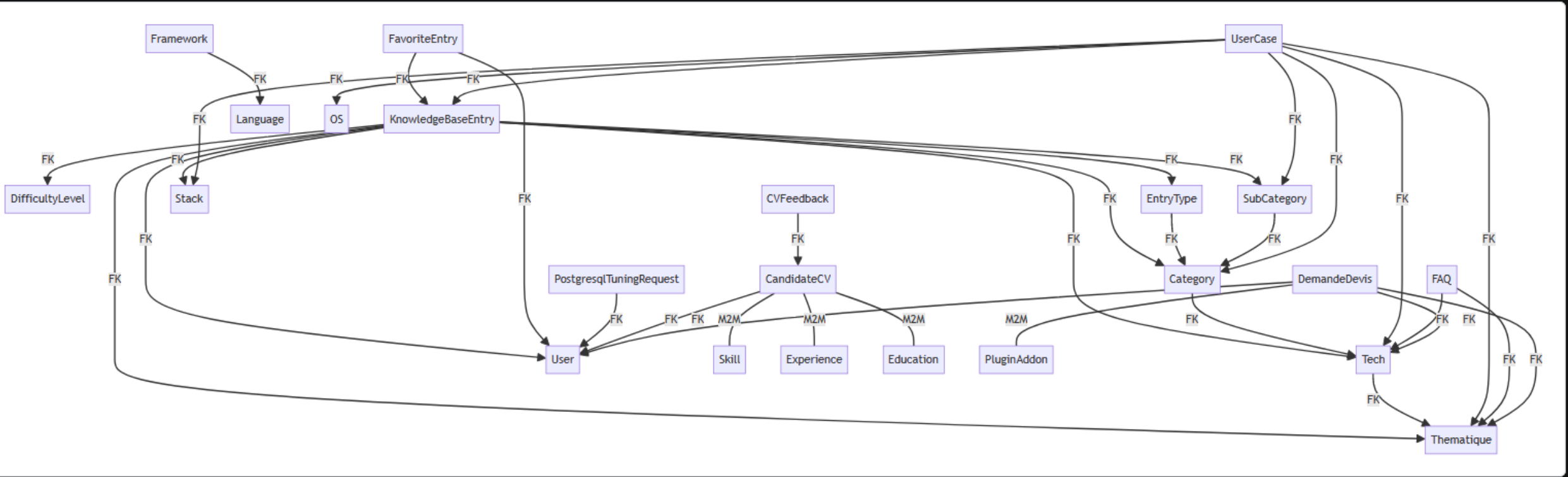
Exporter en .dot Exporter en SVG

Graphe des relations entre modèles

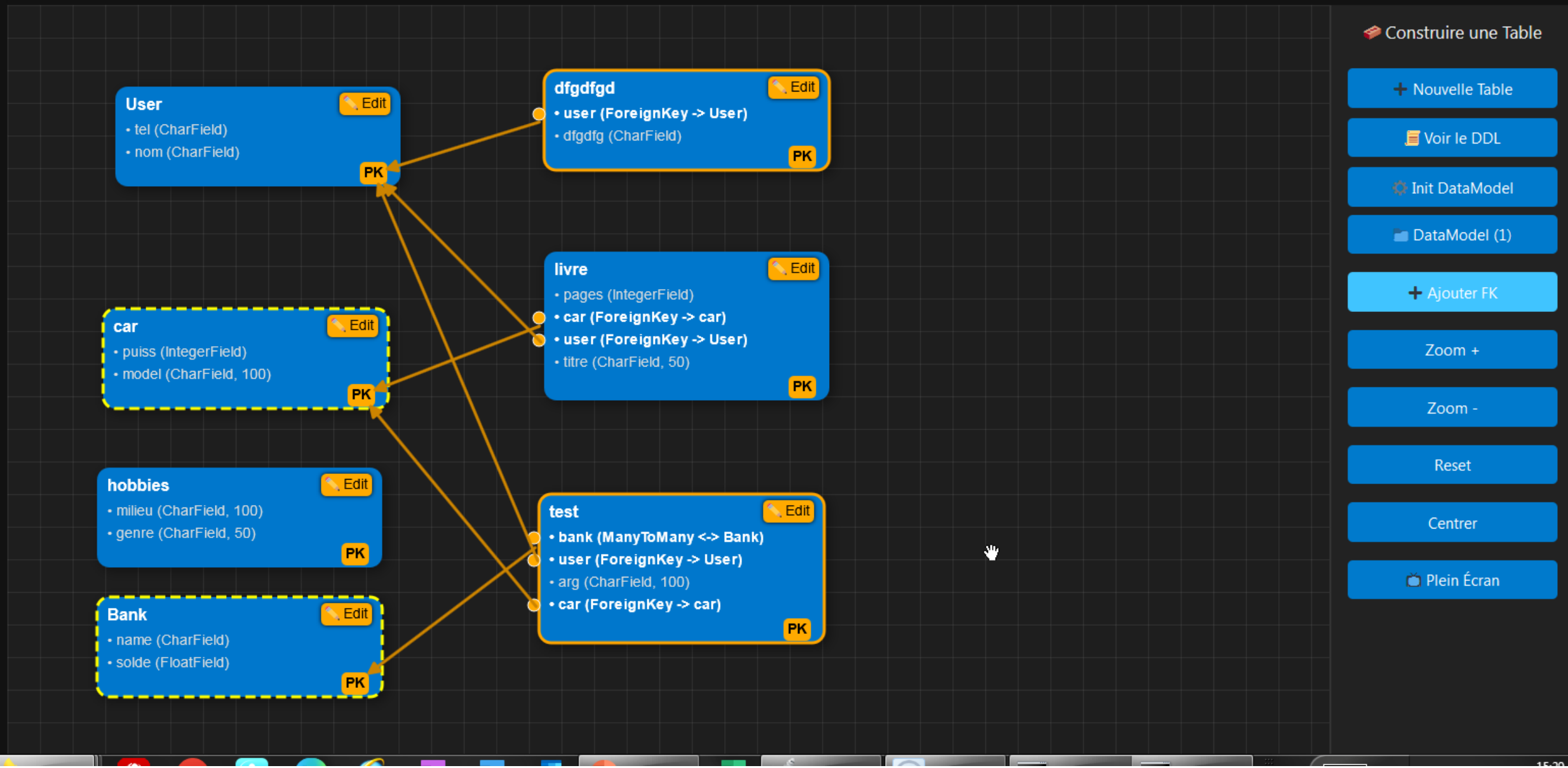


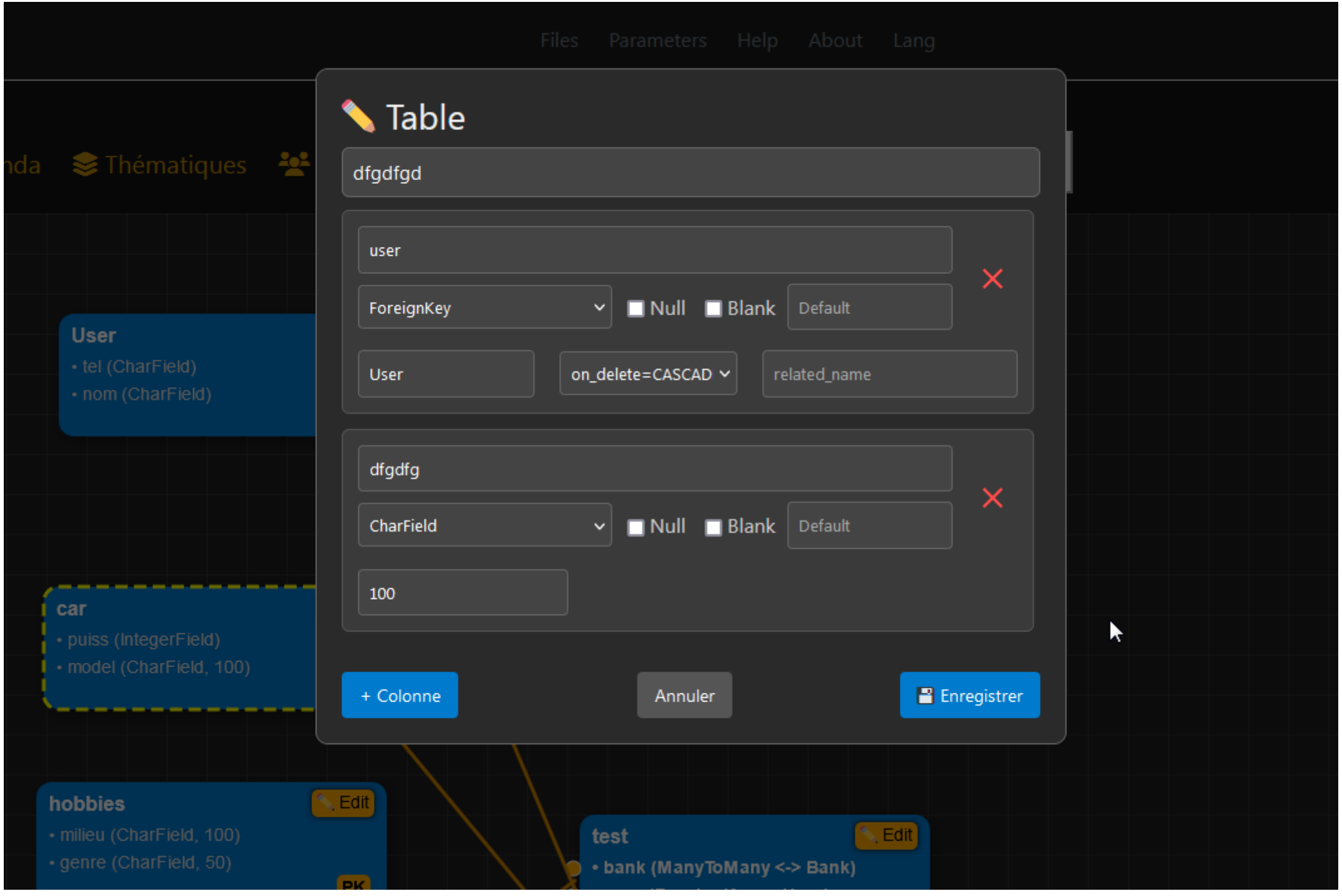
Exporter en .dot Exporter en SVG

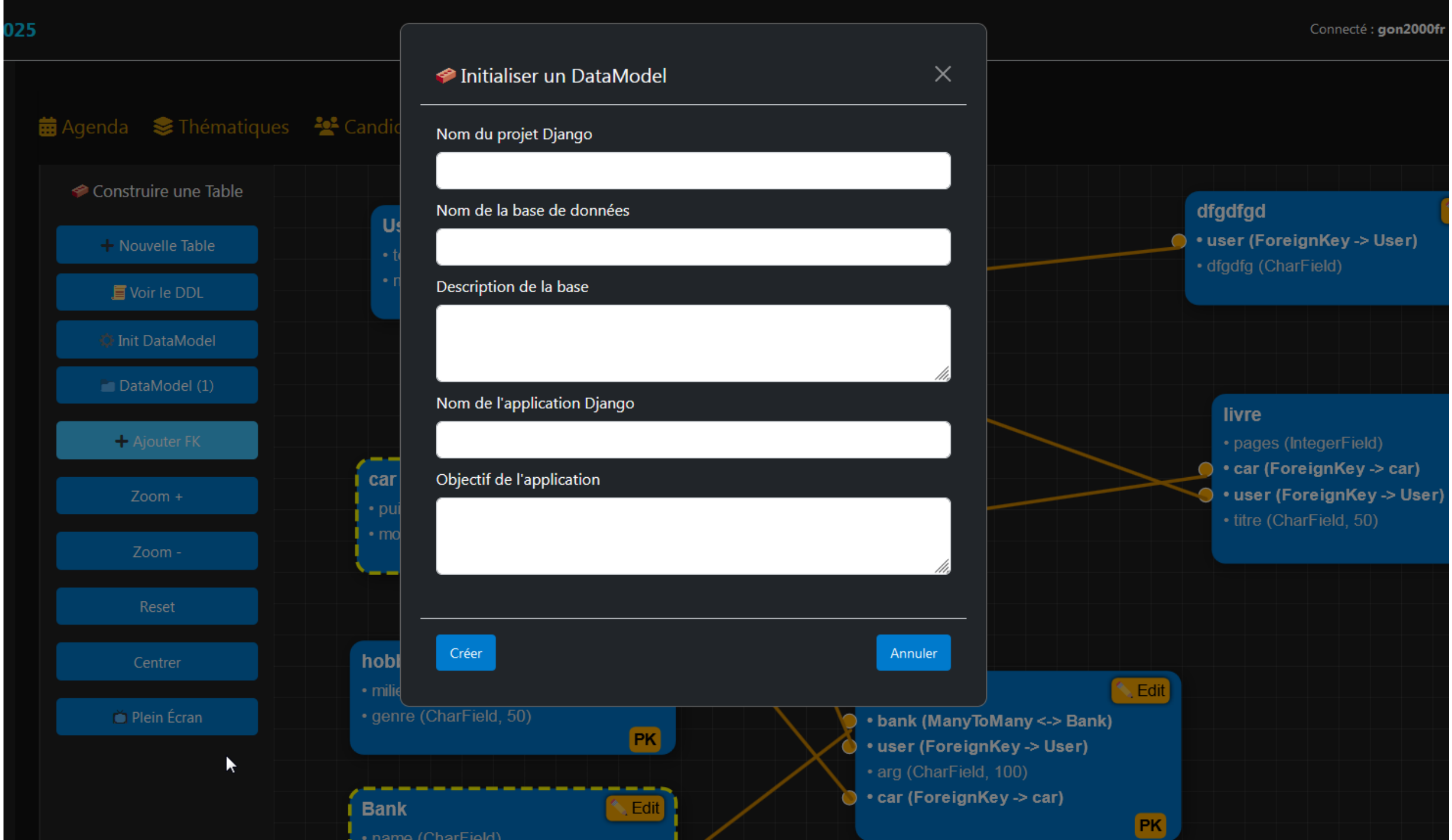
Graphe des relations entre modèles



✅ Le fichier a bien été uploadé et analysé.
22 modèle(s) détecté(s), 169 champ(s) extraits.


User
Edit
• tel (CharField)
• nom (CharField)
PK
dfgdfgd
Edit
• user (ForeignKey -> User)
• dfgdfg (CharField)
PK
livre
Edit
• pages (IntegerField)
• car (ForeignKey -> car)
• user (ForeignKey -> User)
• titre (CharField, 50)
PK
car
Edit
• puiss (IntegerField)
• model (CharField, 100)
PK
hobbies
Edit
• milieu (CharField, 100)
• genre (CharField, 50)
PK
test
Edit
• bank (ManyToMany <-> Bank)
• user (ForeignKey -> User)
• arg (CharField, 100)
• car (ForeignKey -> car)
PK
Bank
Edit
• name (CharField)
• solde (FloatField)
PK
Construire une Table
Nouvelle Table
Voir le DDL
Init DataModel
DataModel (1)
Ajouter FK
Zoom +
Zoom -
Reset
Centrer
Plein Écran


Files
Parameters
Help
About
Lang
Thématiques
Table
dfgdfgd
user
ForeignKey
Null
Blank
Default
User
on_delete=CASCAD
related_name
dfgdfg
CharField
Null
Blank
Default
100
+ Colonne
Annuler
Enregistrer
User
• tel (CharField)
• nom (CharField)
car
• puiss (IntegerField)
• model (CharField, 100)
hobbies
Edit
• milieu (CharField, 100)
• genre (CharField, 50)
test
Edit
• bank (ManyToMany <-> Bank)


025
Connecté : gon2000fr
Agenda
Thématiques
Candid
Construire une Table
Nouvelle Table
Voir le DDL
Init DataModel
DataModel (1)
Ajouter FK
Zoom +
Zoom -
Reset
Centrer
Plein Écran
Initialiser un DataModel
Nom du projet Django
Nom de la base de données
Description de la base
Nom de l'application Django
Objectif de l'application
Créer
Annuler
dfgdfgd
• user (ForeignKey -> User)
• dfgdfg (CharField)
livre
• pages (IntegerField)
• car (ForeignKey -> car)
• user (ForeignKey -> User)
• titre (CharField, 50)
car
hobl
• genre (CharField, 50)
PK
Edit
• bank (ManyToMany <-> Bank)
• user (ForeignKey -> User)
• arg (CharField, 100)
• car (ForeignKey -> car)
PK
Bank
Edit

Agenda Thématiques Candidats Profil Feedback Formations Theme

Réservation d'un créneau (Meeting / Training)

Choisir une date :

jj / mm / aaaa

Créneaux disponibles :

Nom complet :

Email :

Sujet :

Réserver

Ajouter / Mettre à jour votre CV

Dernier CV enregistré :

Preview du CV

Titre du CV

Ex: CV Backend - Juin 2025

Glissez / déposez votre CV ici ou cliquez pour le sélectionner

Parcourir un fichier

Lancer l'analyse du CV