veille_flask_extended

September 29, 2025

1 Veille technologique – Flask (2025)

1.1 1. Flask 3.1.1 – Correctif de sécurité CVE-2025-47278

Release Flask 3.1.1

Cette mise à jour corrige la vulnérabilité CVE-2025-47278 (CWE-683: Function Call With Incorrect Order of Arguments).
Le problème venait d'une construction inversée de la liste des clés de repli (`SECRET_KEY_FALLBACKS`), pouvant entraîner l'utilisation de clés de signature périmées pour les sessions.

Importance : toujours maintenir ses dépendances à jour pour éviter des failles critiques.

1.2 2. Changement majeur dans SQLAlchemy 2.0 et Flask-SQLAlchemy

Documentation Flask-SQLAlchemy – Queries

L'ancienne API `Query` est désormais "legacy".
Les requêtes doivent être créées de manière plus explicite avec `select()` ou `db.select()`, et les objets ORM sont récupérés via `Session.scalars(query)`.

Cela améliore la lisibilité mais nécessite d'adapter le code existant.

1.3 3. Concept de Dockerisation pour Flask

La conteneurisation est essentielle pour la cohérence, la portabilité et la scalabilité.
- Création d'un **Dockerfile** basé sur une image `python:slim`.
- Ajout de dépendances avec `requirements.txt`.
- Utilisation de **Docker Compose** pour gérer des services multi-conteneurs (par ex. Flask + PostgreSQL avec volumes montés pour le dev).

Permet un déploiement reproductible et simplifié.

1.4 4. Flask-RESTful et Sécurité d'API

Guide REST API avec Flask

Flask-Restful simplifie la création d'APIs RESTful via la classe `Resource` (méthodes GET, POST, PUT, DELETE).
Sécurité : utiliser `flask-httpauth` ou `Flask-Security` pour mettre en place une authentification

basique avec `@auth.login_required`.
La sécurité est un enjeu crucial pour toute API exposée.

1.5 5. Gestion des Dépendances et Tests Unitaires

Les discussions de la communauté rappellent :
- Fixer les dépendances (`Flask>=2.3.0,<3.0.0`) pour éviter que des changements mineurs ne cassent des extensions comme Flask-Login.
- Implémenter des tests unitaires : vérifier les codes de réponse (200, 404…) et le format (`application/json`).
Cela renforce la fiabilité et la maintenabilité des projets.

1.6 6. Avantages/Inconvénients de Flask vs FastAPI (2025)

Comparatif frameworks backend
Flask : léger, flexible, parfait pour le prototypage et les microservices.
FastAPI (2025) : plus performant, support natif ASGI/async, validation des types avec **Pydantic**, auto-docs intégrées.
Flask reste pertinent pour les projets légers, mais FastAPI domine pour les APIs modernes et performantes.

1.7 7. Techniques de Développement Avancées (Jinja2 et Décorateurs)

Flask s'appuie sur **Jinja2** pour créer dynamiquement des pages HTML.
Les **décorateurs** (`@app.route`) transforment une fonction Python en route web.
Possibilité de créer des décorateurs personnalisés pour ajouter des fonctionnalités comme le suivi de l'exécution, la surveillance des cookies ou l'analyse des adresses IP.

---

2 Conclusion

- **Sécurité** : mise à jour indispensable (CVE-2025-47278).
- **Migration** : SQLAlchemy 2.0 impose `select()` et `Session.scalars()`.
- **Déploiement** : Dockerisation simplifie portabilité et devOps.
- **API** : Flask-RESTful reste utile mais nécessite des sécurisations.
- **Qualité** : gestion stricte des dépendances + tests unitaires.

- **Comparaison** : Flask reste adapté aux microservices, FastAPI domine pour les APIs modernes.

- **Techniques avancées** : usage de Jinja2 et décorateurs pour enrichir les applications.