Fission : Le Framework Serverless Natif pour Kubernetes

Laura NGUEGANG, Scott DOUANLA, Ange Rody N'Guessan

06 Decembre 2025

Résumé

Fission.io est un *framework open-source* **FaaS** (*Functions as a Service*) conçu pour exécuter des fonctions *serverless* directement et nativement sur n'importe quel cluster **Kubernetes** existant. Son objectif principal est de maximiser la **productivité des développeurs** en masquant les complexités de l'orchestration, leur permettant de se concentrer uniquement sur leur code. Fission résout le problème du *cold start* grâce à son mécanisme de **Pool de Conteneurs Chauds** (*Warm Containers*), qui maintient des *runtimes* de langage pré-chargés pour assurer une haute performance. Complètement agnostique au *cloud*, Fission offre une alternative puissante aux solutions *serverless* propriétaires, tout en héritant de la résilience et du *scaling* de Kubernetes.

Abstract

Fission.io is an **open-source FaaS** (*Functions as a Service*) framework designed to run *serverless* functions directly and natively on any existing **Kubernetes** cluster. Its primary goal is to maximize **developer productivity** by masking orchestration complexities, allowing developers to focus solely on their code. Fission addresses the *cold start* issue through its **Warm Container Pool** mechanism, which maintains pre-loaded language runtimes to ensure high performance. Completely *cloud-agnostic*, Fission offers a powerful alternative to proprietary *serverless* solutions, while inheriting the resilience and scaling capabilities of Kubernetes.

Keywords: Serverless, Kubernetes, FaaS, Fission, Developer Productivity

1. Introduction

Les plateformes **Serverless** (sans serveur) ont transformé la manière dont les applications sont développées et déployées, offrant une mise à l'échelle automatique et une facturation à l'usage.

Cependant, l'adoption de solutions *serverless* des grands fournisseurs de *cloud* entraîne souvent un **verrouillage technologique** (*Vendor Lock-in*), obligeant les développeurs à utiliser des outils propriétaires pour l'intégration et le déploiement.

Dans ce contexte, **Kubernetes** est devenu le standard *de facto* pour l'orchestration des conteneurs. L'émergence de *frameworks* comme Fission.io vise à combler le fossé en permettant aux organisations de bénéficier des avantages du *serverless* tout en conservant leur infrastructure sur Kubernetes, évitant ainsi le *Vendor Lock-in*.

2. Présentation Générale de Fission

Fission.io est un projet *open-source* conçu spécifiquement pour étendre les fonctionnalités de Kubernetes afin d'y intégrer nativement le *Functions as a Service* (FaaS).

2.1. Philosophie et Architecture

L'objectif principal de Fission est de simplifier l'expérience du développeur. Contrairement à un déploiement Kubernetes traditionnel qui requiert la gestion de **Dockerfiles**, d'images, de *Deployments* et de *Services*, Fission abstrait toutes ces complexités.

Fission s'appuie sur trois concepts clés :

- **Fonction (Function) :** L'unité de code métier (ex. : un script Python ou une méthode Java) fournie par le développeur.
- **Environnement (Environment) :** Une image conteneur réutilisable qui fournit le *runtime* du langage (NodeJS, Python, Java, Go, etc.). Fission utilise ces environnements pour exécuter les fonctions.
- **Déclencheur (Trigger) :** L'événement qui invoque la fonction (requête HTTP, timer Cron, message Kafka, événement Kubernetes, etc.).

2.2. Haute Performance et Cold Start

L'un des principaux défis du *serverless* est le temps de **Cold Start** (démarrage à froid), qui peut entraîner une latence inacceptable pour certaines API.

Fission résout ce problème en utilisant le **Pool de Conteneurs Chauds** (*Warm Containers*). Fission maintient un ensemble de *Pods* Kubernetes déjà démarrés pour chaque *Environnement*. Ces *Pods* ont le *runtime* du langage chargé et sont en attente.

Lorsqu'une fonction est appelée :

1. Fission prend un conteneur chaud disponible.
2. Il y injecte **instantanément** le code de la fonction.
3. La fonction s'exécute avec une latence minimale (souvent < 100 ms), assurant une haute performance.

3. Avantages pour la Productivité du Développeur

Le *workflow* de Fission est optimisé pour maximiser la vitesse de développement :

- **Workflow Simplifié :** Le développeur n'a besoin que de son code et d'un outil CLI simple (`fission fn create`, `fission route create`). La construction d'images Docker, la gestion des *Pods* et les *manifests* Kubernetes sont gérés automatiquement en arrière-plan.
- **Agnosticisme du Cloud :** Fission permet aux équipes de conserver leur architecture *serverless* sur leur infrastructure privée ou de la migrer facilement entre différents fournisseurs de *cloud*, éliminant le risque de *Vendor Lock-in*.
- **Intégration Kubernetes :** Les fonctions Fission peuvent interagir de manière transparente avec d'autres services, bases de données ou ressources gérées par Kubernetes, favorisant une architecture unifiée.

4. Conclusion

Fission.io représente une solution Serverless mature et performante pour les entreprises qui ont déjà adopté Kubernetes. En se concentrant sur le code et en automatisant les processus d'orchestration et de mise à l'échelle, Fission atteint son objectif de maximiser la productivité des développeurs tout en garantissant des temps de réponse rapides et une portabilité totale. C'est une alternative puissante et *open-source* aux offres FaaS des géants du *cloud*.

5. References

Références commentées

- Documentation Officielle Fission.io [1] : Décrit les concepts de Functions, Environments et Triggers qui composent l'architecture Serverless de Fission, ainsi que son fonctionnement.
- Fission.io : Performance et Cold Start [2] : Explique en détail comment le mécanisme de Pool de Conteneurs Chauds permet d'atteindre une latence de *cold start* de l'ordre de 100 millisecondes, un avantage clé de Fission.

References

[1] Fission Contributors. *Fission Documentation: Concepts and Architecture*. URL: `https://fission.io/docs/concepts/` (ou un lien similaire) [2] Fission Contributors. *Performance: 100msec cold start*. URL: `https://fission.io/docs/` (rechercher l'article sur la performance) [3] [Nom de l'auteur]. *Why Kubernetes is the Right Place for Serverless*. (Article de blog ou Medium si trouvé) [4] Fission Contributors. *fission/fission: Fast and Simple Serverless Functions for Kubernetes*. URL: `https://github.com/fission/fission` [5]

[Nom(s) de l'auteur(s)]. *Mitigating Cold Start Problem in Serverless Computing: A Reinforcement Learning Approach*. (Ou un article pertinent sur le *cold start* et les techniques de *pre-warming*).