PRÉSENTATION VEILLE TECHNOLOGIQUE

THÈME

fission

Kubernetes-native Serverless Framework

MEMBRES DU GROUPE :
Laura NGUEGANG
SCOTT DOUANLA
N'GUESSAN DJESSE ANGE

SOMMAIRE

INTRODUCTION

fission

Qu'est-ce que Fission et pourquoi il existe ?

- Framework open-source pour le serverless sur Kubernetes
- Créé par Platform9 en 2016, maintenu activement
- Permet de déployer des fonctions sans gérer l'infrastructure sous-jacente
- Écrit en Go pour des performances optimales

Philosophie "Write code, not infrastructure"

- Focus sur la logique métier uniquement
- Abstraction complète de l'infrastructure
- Déploiement en quelques secondes

Problème sans Fission

Déploiement classique sur Kubernetes

- Écriture de Dockerfiles complexes
- Configuration de Deployments, Services, Ingress
- Gestion manuelle du scaling
- Mise à jour et rollback compliqués
- Temps de déploiement : 5-10 minutes

```yaml
# 100+ lignes de YAML pour une fonction simple
apiVersion: apps/v1
kind: Deployment
...
apiVersion: v1
kind: Service
...
```

Coût en complexité : Courbe d'apprentissage importante / Maintenance lourde / Erreurs de configuration fréquentes

INTRODUCTION

fission

Problème résolu avec Fission

```
fission function create --name hello \
  --env python --code hello.py
```

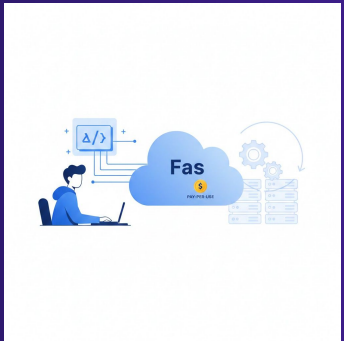
Résultat
- Déploiement en < 1 seconde - Scaling automatique configuré
- Monitoring intégré
- Pas de YAML à écrire

Gain
- 95% de code en moins
- Déploiement 100x plus rapide - Focus sur la valeur business

CONCEPTS CLÉS

fission

Architecture FaaS (Function as a Service)



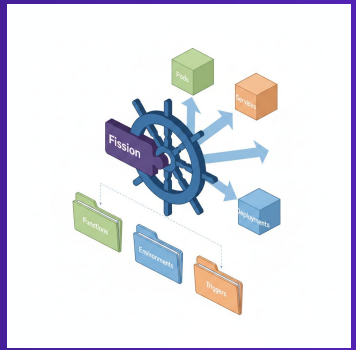
Principe

- Vous écrivez uniquement la logique métier
- L'infrastructure scale automatiquement
- Paiement à l'utilisation

Avantages

Pas de gestion de serveurs
Focus sur le code
Optimisation des ressources

Intégration Kubernetes



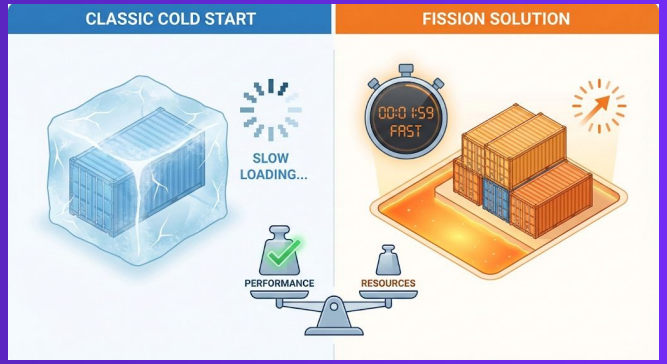
Fission = Kubernetes-native

Utilise les primitives K8s (Pods, Services, Deployments)
Installation via Helm
Exploitation du scaling natif de K8s

Custom Resource Definitions (CRDs)

Functions
Environments
Triggers

Problème du Cold Start



Cold Start classique

Démarrage d'un container : plusieurs secondes
Mauvaise expérience utilisateur
Latence imprévisible

Solution Fission

Pool de containers pré-chauffés
Temps de réponse < 100ms
Trade-off : performance vs ressources

FONCTIONNEMENT TECHNIQUE

fission

Composants principaux

Controller

API server de Fission
Gère le CRUD des fonctions, triggers, environnements
Stocke la configuration dans etc (via K8s CRDs)

Router

Point d'entrée pour les requêtes HTTP
Fait le mapping requête → fonction
Gère le load balancing vers les pods de fonctions

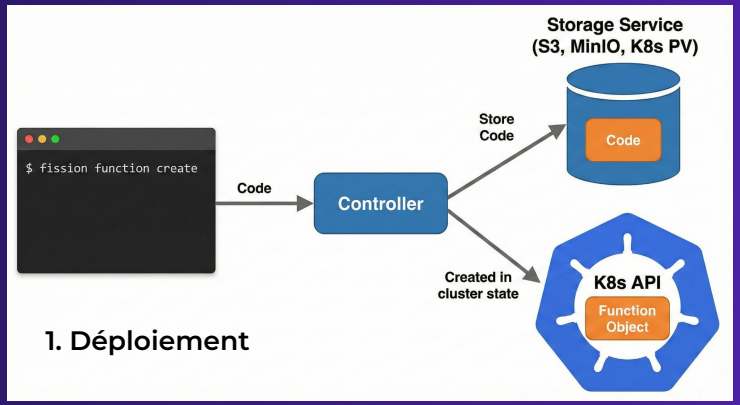
Executor

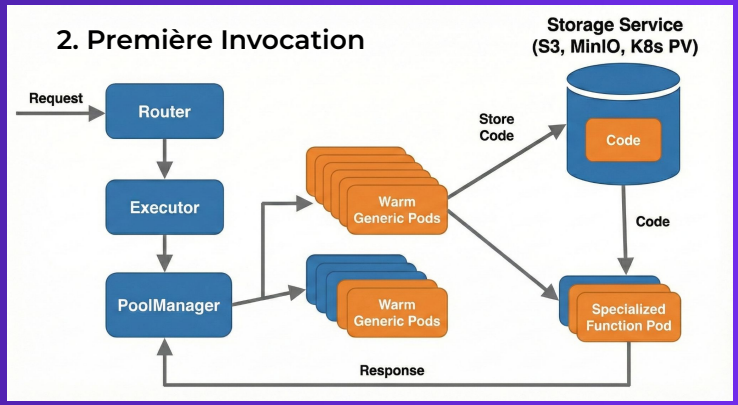
Responsable du cycle de vie des pods de fonctions
Trois types d'executors :

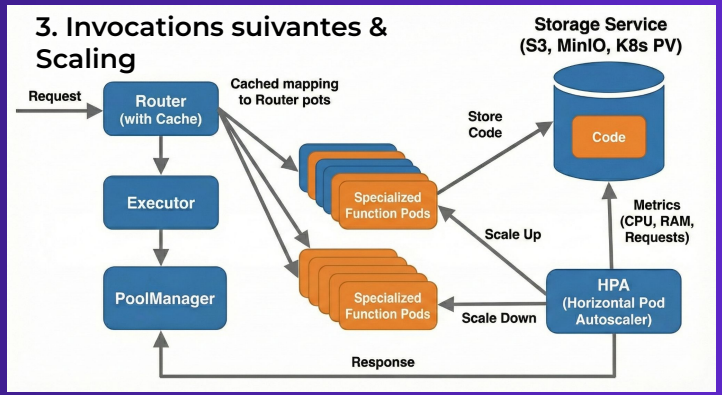
- PoolManager : pool de containers chauds (rapide, consomme plus)
- NewDeploy : crée un deployment K8s par fonction (scalable, cold start)
- Container : conteneur unique partagé (économique, moins isolé)

FONCTIONNEMENT TECHNIQUE

Cycle de vie d'une fonction

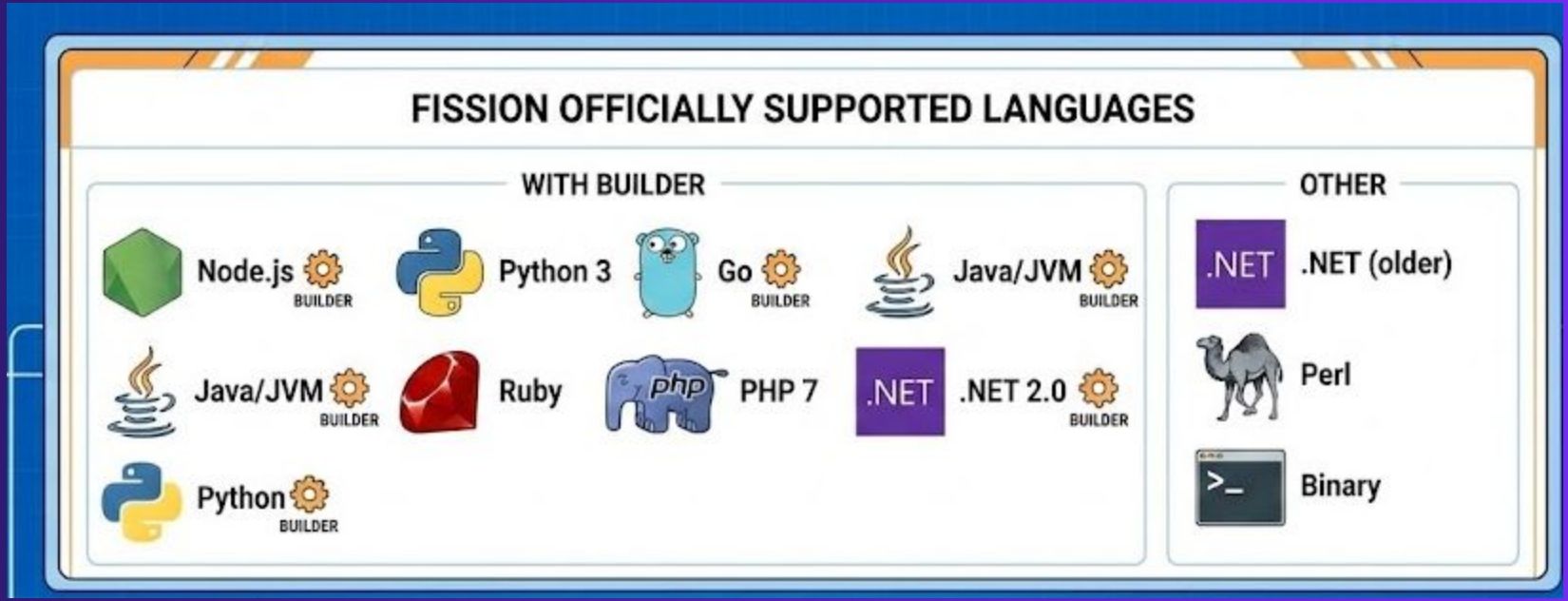






FONCTIONNEMENT TECHNIQUE

Environnements d'exécution supportés



DEMO / EXEMPLE

CONCLUSION