PLAN D'ARCHITECTURE - PLATEFORME MECAPY

🏗️ ARCHITECTURE PROPOSÉE : Hybrid Task Orchestration System

📊 Contexte et contraintes

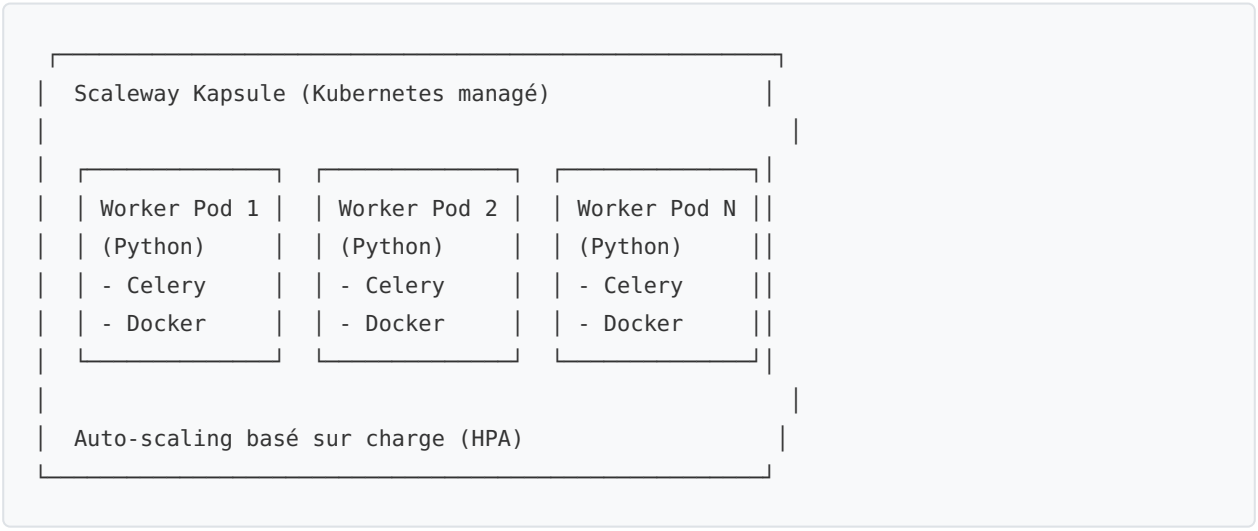
- ✗ Serverless Functions limitées (1000 max chez Scaleway)
- ✓ Provider français/européen souhaité
- ✓ Stack actuelle : FastAPI, Keycloak, Scaleway Object Storage
- ✓ Besoin : Calculs courts (secondes) ET longs (heures/jours)
- ✓ Multi-tenant avec isolation sécurisée

🎯 SOLUTION : Architecture Hybride à 3 Niveaux

Niveau 1 : API Gateway & Orchestration (FastAPI actuelle)

Rôle : - Réception des requêtes utilisateurs - Validation, authentification, autorisation - Dispatch des tâches vers workers - Gestion du cycle de vie (statut, résultats, notifications)

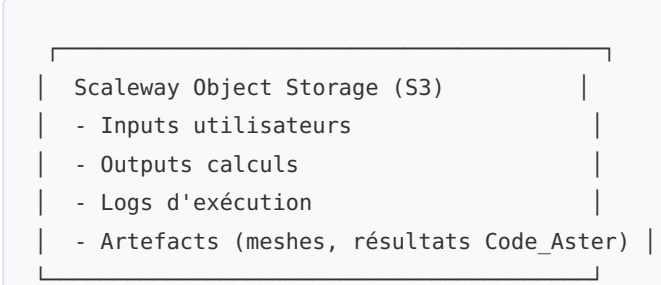
Niveau 2 : Worker Pool Dynamic (Kubernetes sur Scaleway Kapsule)



Technologie recommandée : - **Celery** : Système de task queue distribué (Python-native) - **Redis** : Broker de messages + résultats backend - **Kubernetes HPA** : Auto-scaling horizontal (2-50 workers) - **Docker** : Isolation des tâches utilisateur

Avantages : - ✅ **Scalabilité infinie** (pas de limite de 1000) - ✅ **Workers dynamiques** (scale up/down automatique) - ✅ **Isolation sécurisée** (chaque task dans son conteneur) - ✅ **Provider français** (Scaleway Kapsule = Paris/Amsterdam)

Niveau 3 : Storage & Résultats (Scaleway Object Storage)



🔄 FLUX D'EXÉCUTION D'UN CALCUL

1. Soumission d'une tâche

```
Utilisateur → FastAPI → Validation → Redis Queue
                ↓
          PostgreSQL (métadonnées)
```

2. Traitement par worker

```
 Celery Worker poll Redis → Pull inputs (S3)
    ↓
Exécution Python/Code_Aster (Docker isolé)
    ↓
Push résultats (S3) → Update PostgreSQL (status: completed)
    ↓
Notification utilisateur (webhook/SSE)
```

3. Récupération résultats

```
Utilisateur → FastAPI → PostgreSQL (métadonnées)
                ↓
        S3 (résultats signés URL)
```

COMPOSANTS TECHNIQUES DÉTAILLÉS

A. File d'attente : Redis + Celery

Pourquoi Celery ? - ✅ Natif Python (intégration simple avec FastAPI) - ✅ Supporte tâches courtes ET longues - ✅ Priorités de tâches (fast-lane pour calculs courts) - ✅ Retry automatique, timeout, rate limiting - ✅ Monitoring avec Flower (UI web)

Configuration Redis : - **Redis Managed Database** (Scaleway) : €10/mois pour débuter - **3 queues** : - `high_priority` : calculs < 5s (FIFO strict) - `default` : calculs < 5min - `long_running` : calculs > 5min (throttled)

B. Workers : Kubernetes sur Scaleway Kapsule

Setup recommandé :

```
 # Deployment YAML simplifié
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mecapy-worker
spec:
  replicas: 3  # Auto-scale 2-50
  template:
    spec:
      containers:
      - name: celery-worker
        image: mecapy/worker:latest
        resources:
          requests:
            cpu: "500m"
            memory: "1Gi"
          limits:
            cpu: "2000m"
            memory: "4Gi"
        env:
        - name: CELERY_BROKER_URL
          value: "redis://redis.scaleway.com:6379/0"
```

Auto-scaling basé sur : - CPU > 70% → Scale up - Queue depth > 100 messages → Scale up - CPU < 30% pendant 5min → Scale down

Coût estimé : - **Kapsule** : Gratuit (contrôle plane) - **Nodes** : 3x DEV1-M (4 vCPU, 8GB) = €0.024/h/node × 3 = ~€52/mois - Scale up jusqu'à 10 nodes lors des pics

---

C. Isolation des calculs : Docker-in-Docker (DinD)

Architecture de sécurité :

```
Worker Pod
  └─ Celery Process
      └─ Docker Container (task execution)
          - CPU limits (cgroups)
          - Memory limits (cgroups)
          - Network isolation (no internet)
          - Read-only filesystem (sauf /tmp)
          - Timeout strict (kill après X min)
```

Validation pré-exécution : 1. **Analyse statique** du code (Bandit, safety checks) 2. **Sandbox Python** avec `RestrictedPython` 3. **Whitelist d'imports** (numpy, scipy, etc.) 4. **Pas d'accès filesystem** (sauf montage S3 read-only)

D. Base de données : PostgreSQL (Scaleway)

Tables principales :

```
 -- Tasks metadata
CREATE TABLE tasks (
    id UUID PRIMARY KEY,
    user_id UUID NOT NULL,
    name VARCHAR(255),
    status ENUM('queued', 'running', 'completed', 'failed'),
    priority INTEGER,
    runtime_seconds INTEGER,
    created_at TIMESTAMP,
    started_at TIMESTAMP,
    completed_at TIMESTAMP,
    result_s3_key VARCHAR(512),
    error_message TEXT
);

-- Workflows
CREATE TABLE workflows (
    id UUID PRIMARY KEY,
    user_id UUID NOT NULL,
    name VARCHAR(255),
    dag JSON,  -- Graph de dépendances
    status ENUM('draft', 'running', 'completed', 'failed')
```

```
);

-- Studies
CREATE TABLE studies (
    id UUID PRIMARY KEY,
    company_id UUID,
    name VARCHAR(255),
    workflows JSON[]
);
```

Coût : ~€18/mois pour PostgreSQL 1GB (Scaleway DB)

🏛️ ARCHITECTURE ALTERNATIVE (pour très gros volumes)

Si vous dépassez 10 000 calculs/jour, considérer :

Option B : Scaleway Kubernetes + Serverless Containers

```
 API FastAPI → Kubernetes Jobs (vs Workers permanents)
    ↓
Chaque tâche = 1 Kubernetes Job éphémère
    ↓
Auto-cleanup après exécution
```

Avantages : - ✅ Pas de workers idle (coût = 0 quand inactif) - ✅ Isolation parfaite (1 pod = 1 tâche) - ✅ Scale à l'infini (limité par quota Kapsule)

Inconvénients : - ❌ Overhead plus important (30s startup vs 1s avec Celery) - ❌ Plus complexe à monitorer

---

📊 COMPARAISON DES SOLUTIONS

Critère	Serverless Functions	Celery + K8s Workers	K8s Jobs
Limite	1000 functions	∞ (limité par nodes)	∞
Coût base	€0 si inactif	~€50/mois (3 workers)	~€50/mois (nodes)
Coût à grande échelle	€€€ (pay-per-invocation)	€€ (nodes only)	€€ (nodes only)
Latence démarrage	~500ms	~50ms (worker warm)	~30s (pod spawn)
Isolation	✔✔✔	✔✔ (Docker)	✔✔✔
Monitoring	Fragmented	✔ Flower + Grafana	K8s dashboard
Complexité	Simple	Moyenne	🟠 Complexe
Provider FR	✔ Scaleway	✔ Scaleway	✔ Scaleway

🎯 RECOMMANDATION FINALE

Phase 1 (MVP) : Celery + Redis + Docker Workers

- **Pourquoi** : Simple, scalable, coût prévisible
- **Stack** :
- FastAPI (API) → Clever Cloud
- Redis (queue) → Scaleway Managed Redis
- Celery Workers → 3x VMs Scaleway (DEV1-M)
- PostgreSQL → Scaleway DB
- Object Storage → Scaleway S3
- **Coût estimé** : ~€80/mois pour 1000 calculs/jour

Phase 2 (Scale) : Migration vers Kubernetes

- Quand > 5000 calculs/jour
- Déployer Celery workers dans Kapsule
- Auto-scaling HPA
- Coût optimisé (~€150/mois pour 10 000 calculs/jour)

Phase 3 (Enterprise) : Hybrid Architecture

- Calculs courts (< 5s) → Serverless Containers (Scaleway)
- Calculs longs → Kubernetes Workers
- Meilleur rapport coût/performance

🔧 SERVICES SCALEWAY RECOMMANDÉS

Service	Usage	Prix estimé
Kapsule (K8s)	Worker orchestration	Gratuit (control plane)
Instance DEV1-M	Worker nodes (3x)	€52/mois
Managed Redis	Task queue	€10/mois
Managed PostgreSQL	Metadata	€18/mois
Object Storage	Results/inputs	€0.01/GB (~€10/mois)
Load Balancer	API HA	€8/mois
Total		**~€98/mois**

🚀 PLAN DE MIGRATION

Étape 1 : Setup Infrastructure (Semaine 1-2)

1. Provisionner Redis Managed Database
2. Configurer PostgreSQL avec schéma
3. Déployer 3x VMs avec Docker + Celery

Étape 2 : Intégration API (Semaine 3-4)

1. Ajouter routes FastAPI pour task submission
2. Intégrer Celery avec FastAPI
3. Implémenter workers basiques (Python tasks)

Étape 3 : Sécurisation (Semaine 5-6)

1. Isolation Docker-in-Docker
2. Resource limits (CPU/RAM)
3. Validation code statique

Étape 4 : Workflows (Semaine 7-8)

1. Orchestration multi-tâches avec Celery Canvas
2. DAG execution avec dépendances
3. Retry logic et error handling

Étape 5 : Monitoring (Semaine 9)

1. Flower dashboard (Celery)
2. Prometheus + Grafana (metrics)
3. Alerting (PagerDuty/email)

✅ AVANTAGES DE CETTE ARCHITECTURE

1. **Pas de limite artificielle** (1000 functions)
2. **Coût linéaire** (scale selon usage)
3. **Provider français** (RGPD-friendly)
4. **Tech mature** (Celery = production-proven)
5. **Flexibilité** (Python natif, facile à étendre)
6. **Isolation** (Docker par tâche)
7. **Monitoring** (Flower + Grafana)
8. **Support long-running** (heures/jours)

Document généré le : 2025-09-30 **Version** : 1.0 **Auteur** : Architecture Team MecaPy