ARCHITECTURE SERVERLESS SIMPLIFIÉE - MECAPY

🎯 OBJECTIF : Zéro gestion d'infrastructure

Contraintes

- ✅ Développeur solo (pas de DevOps)
- ✅ Provider français/européen
- ✅ Focus sur le produit, pas l'infra
- ✅ Calculs longs = problème futur (si besoin)

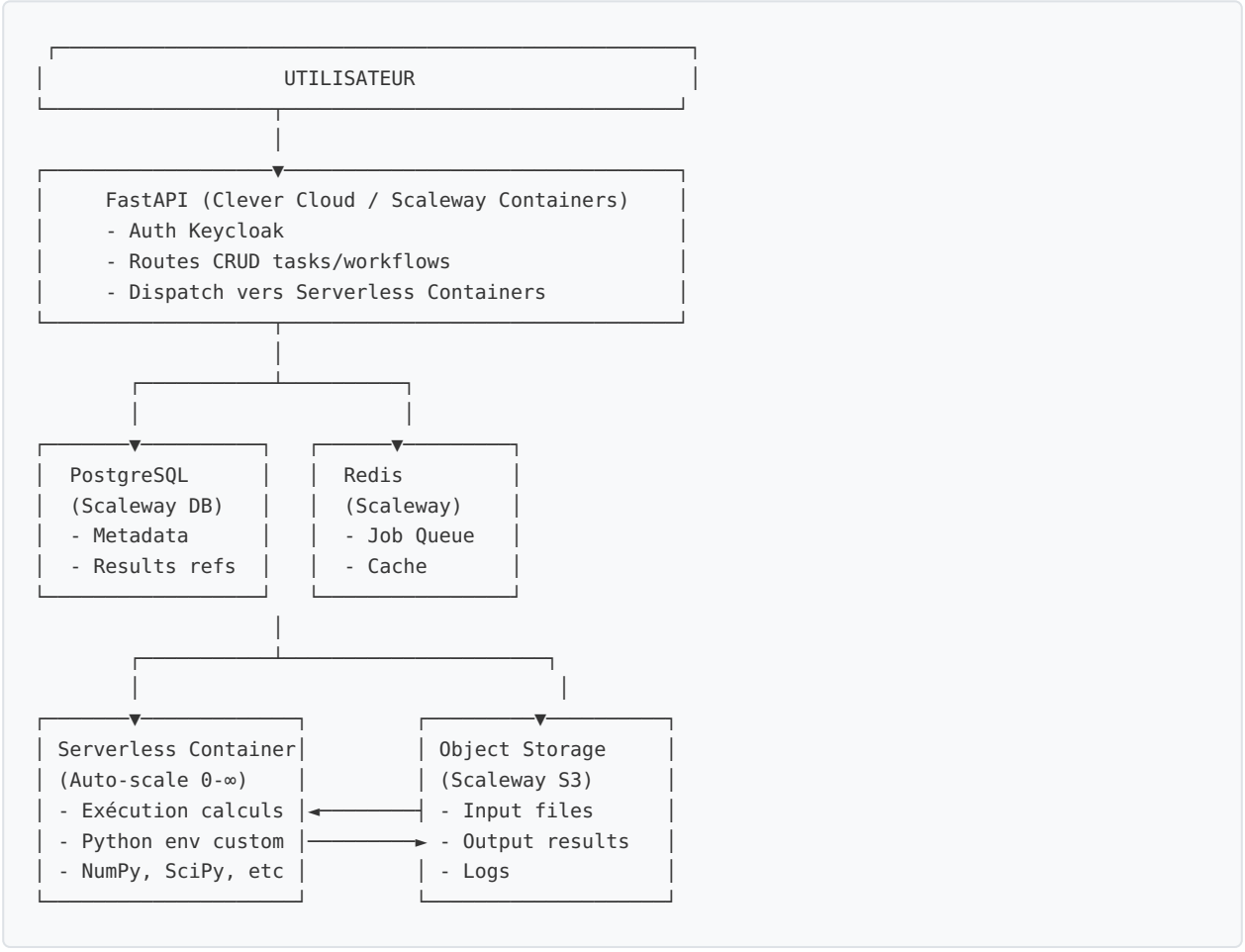
🚀 SOLUTION RECOMMANDÉE : Scaleway Serverless Containers

Pourquoi Serverless Containers > Serverless Functions ?

Critère	Functions	Containers
Limite nombre	1000 max	∞ (illimité)
Durée exécution	15min max	10min par défaut
Flexibilité	Code ZIP only	Image Docker custom
Dépendances	Limitées	Full control
Cold start	~500ms	~2-3s
Coût idle	€0	€0
Gestion	ϟ Zero	ϟ Minimal

✅ Verdict : Serverless Containers = meilleur choix

🏗️ ARCHITECTURE PROPOSÉE (100% Serverless)



📋 COMPOSANTS DÉTAILLÉS

1. API FastAPI (Clever Cloud ou Scaleway Containers)

Option A : Clever Cloud (recommandé pour MVP)

```
 Type: Python app
Région: Paris
Auto-scaling: 1-3 instances
Coût: ~€7-20/mois
```

Option B : Scaleway Serverless Containers (plus scalable)

```
 Type: Container
Image: registry.scaleway.com/mecapy/api:latest
Scaling: 0-10 instances
Coût: Pay-per-use (€0 si inactif)
```

Endpoints :

```
 POST /api/tasks              # Créer une tâche
POST /api/tasks/{id}/execute # Exécuter (dispatch to container)
GET  /api/tasks/{id}/status  # Polling status
GET  /api/tasks/{id}/results # Récupérer résultats
```

---

2. Serverless Containers (workers de calcul)

Configuration :

```
 # Dockerfile pour worker
FROM python:3.12-slim

# Installer dépendances scientifiques
RUN pip install numpy scipy pandas matplotlib

# Copier le code de la plateforme
COPY worker/ /app/
WORKDIR /app

# Endpoint HTTP pour recevoir les jobs
CMD ["uvicorn", "worker:app", "--host", "0.0.0.0", "--port", "8080"]
```

Déploiement :

```
 # Build et push image
docker build -t mecapy-worker:v1 .
docker tag mecapy-worker:v1 registry.scaleway.com/mecapy/worker:v1
docker push registry.scaleway.com/mecapy/worker:v1

# Créer namespace serverless
scw container namespace create name=mecapy region=fr-par

# Déployer container
scw container container create \
  namespace-id=<namespace-id> \
  name=mecapy-worker \
  registry-image=registry.scaleway.com/mecapy/worker:v1 \
  min-scale=0 \
  max-scale=10 \
```

```
  memory-limit=2048 \
  cpu-limit=1000
```

Code worker simplifié :

```
 # worker.py
from fastapi import FastAPI
import boto3
import json

app = FastAPI()

@app.post("/execute")
async def execute_task(payload: dict):
    """
    Reçoit un job de calcul depuis l'API.

    payload = {
        "task_id": "uuid",
        "code": "base64_encoded_python",
        "inputs_s3_key": "inputs/uuid.json",
        "user_id": "uuid"
    }
    """

    # 1. Download inputs from S3
    s3 = boto3.client('s3')
    inputs = s3.get_object(
        Bucket='mecapy-storage',
        Key=payload['inputs_s3_key']
    )

    # 2. Execute user code (sandboxed)
    result = execute_user_code(
        code=payload['code'],
        inputs=json.loads(inputs['Body'].read())
    )

    # 3. Upload results to S3
    s3.put_object(
        Bucket='mecapy-storage',
        Key=f"results/{payload['task_id']}.json",
        Body=json.dumps(result)
    )

    # 4. Return status
    return {"status": "completed", "result_key": f"results/{payload['task_id']}.json"}
```

Limites Scaleway Serverless Containers : - ✅ **Nombre illimité** de containers (pas de limite 1000) - ✅ **Auto-scaling** : 0 → 10 instances automatiquement - ⚠️ **Timeout** : 10min par défaut (peut être augmenté à 15min) - ⚠️ **Cold start** : 2-3s (acceptable pour calculs longs)

---

3. Job Queue : Redis (Scaleway Managed)

Pourquoi Redis ? - ✅ Gérer la file d'attente de tâches - ✅ Éviter surcharge des containers - ✅ Retry automatique en cas d'échec

Configuration :

```
 # api/services/queue.py
import redis
import json

class JobQueue:
    def __init__(self):
        self.redis = redis.from_url(os.getenv("REDIS_URL"))

    def enqueue(self, task_id: str, payload: dict):
        """Ajoute un job dans la queue"""
        self.redis.rpush("mecapy:jobs", json.dumps({
            "task_id": task_id,
            "payload": payload,
            "enqueued_at": datetime.utcnow().isoformat()
        }))

    def dequeue(self) -> dict | None:
        """Récupère le prochain job"""
        job = self.redis.blpop("mecapy:jobs", timeout=5)
        return json.loads(job[1]) if job else None
```

Coût : ~€10/mois (Managed Redis Database 256MB)

---

4. Orchestration des tâches

Flux d'exécution :

```
 # api/routes/tasks.py
from fastapi import APIRouter, BackgroundTasks
import httpx

router = APIRouter()

@router.post("/tasks/{task_id}/execute")
async def execute_task(task_id: str, background_tasks: BackgroundTasks):
    """Lance l'exécution d'une tâche"""

    # 1. Récupérer métadonnées
    task = await db.get_task(task_id)

    # 2. Préparer inputs dans S3
    s3_key = await upload_inputs_to_s3(task.inputs)

    # 3. Enqueue job
```

```
    await queue.enqueue(task_id, {
        "code": task.code,
        "inputs_s3_key": s3_key,
        "user_id": task.user_id
    })

    # 4. Lancer worker async (background)
    background_tasks.add_task(dispatch_to_worker, task_id)

    return {"status": "queued", "task_id": task_id}


async def dispatch_to_worker(task_id: str):
    """Dispatch job vers Serverless Container"""

    # Récupérer job de la queue
    job = await queue.dequeue()

    # Appeler Serverless Container
    async with httpx.AsyncClient() as client:
        response = await client.post(
            "https://mecapy-worker-<namespace>.containers.fnc.fr-par.scw.cloud/execute",
            json=job["payload"],
            timeout=600.0  # 10min max
        )

    # Update task status
    await db.update_task_status(task_id, "completed", response.json())
```

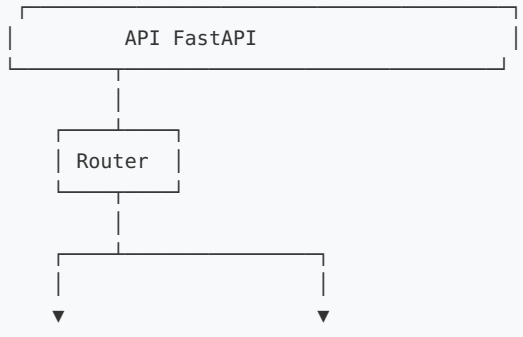
🔥 GESTION DES CALCULS LONGS (> 10min)

Stratégie progressive :

Phase 1 (MVP) : Jusqu'à 10min max

✅ Serverless Containers suffisent ✅ Couvre 95% des cas d'usage initiaux

Phase 2 (Si besoin > 10min) : Architecture hybride



```
Short tasks          Long tasks
(< 10min)            (> 10min)
    |                    |
    ▼                    ▼
Serverless          Dedicated VM
Container           (OVH GPU/CPU)
(Scaleway)          (lancé on-demand)
```

Option pour calculs longs : 1. **Scaleway GPU Instances** (on-demand) - Lancées uniquement pour calculs > 10min - Détruite après exécution - Coût : pay-per-use (~€0.50/h GPU)

1. **OVH Public Cloud Instances** (alternative)
2. Plus flexible pour calculs très longs
3. Support GPU/CPU haute performance

Trigger automatique :

```
 # api/services/task_router.py
def route_task(task: Task) -> ExecutionBackend:
    """Choisir le backend selon estimation"""

    estimated_duration = task.estimate_duration()

    if estimated_duration < 600:  # < 10min
        return "serverless_container"
    else:
        return "dedicated_instance"
```

💰 COÛTS ESTIMÉS (100% Serverless)

Phase 1 : MVP (0-1000 calculs/jour)

Service	Type	Coût mensuel
FastAPI	Clever Cloud (Nano)	€7-10/mois
PostgreSQL	Scaleway DB 1GB	€18/mois
Redis	Scaleway DB 256MB	€10/mois
Object Storage	Scaleway S3 (10GB)	€0.20/mois
Serverless Containers	Pay-per-use (100h/mois)	€5-15/mois

Service	Type	Coût mensuel
Registry	Scaleway Container Registry	Gratuit (100GB)
Total		**~€40-55/mois**

Calculs serverless : - €0.00001/GB-s (mémoire) - €0.000012/vCPU-s - Exemple : 1000 calculs × 30s × 2GB = ~€6/mois

Phase 2 : Scale (1000-10 000 calculs/jour)

Service	Coût mensuel
Base (API + DB + Redis)	€35/mois
Serverless Containers (1000h/mois)	€50-80/mois
Object Storage (100GB)	€2/mois
Total	**~€90-120/mois**

⚡ AVANTAGES DE CETTE ARCHITECTURE

✅ Pour le développeur solo

1. **Zéro gestion d'infra** : Pas de K8s, pas de VMs à maintenir
2. **Scaling automatique** : 0 → 10 containers selon charge
3. **Coût variable** : Paye uniquement l'usage réel
4. **Déploiement simple** : `docker push` + config Scaleway
5. **Monitoring inclus** : Dashboard Scaleway native

✅ Pour les utilisateurs

1. **Latence acceptable** : 2-3s cold start (négligeable si calcul > 10s)
2. **Fiabilité** : Infrastructure managée Scaleway
3. **Scaling transparent** : 100 ou 10 000 calculs, même architecture

✅ Pour l'évolution

1. **Phase 1** : 100% serverless (simple)
2. **Phase 2** : Ajouter VMs on-demand pour calculs longs (si besoin)
3. **Phase 3** : Migration K8s uniquement si > 50 000 calculs/jour

---

🚀 PLAN DE DÉPLOIEMENT (QUICK START)

Semaine 1 : Infrastructure

```
 # 1. Créer compte Scaleway
scw init

# 2. Provisionner services
scw rdb instance create \
  name=mecapy-postgres \
  engine=postgresql-15 \
  node-type=db-dev-s

scw redis cluster create \
  name=mecapy-redis \
  node-type=RED1-micro

# 3. Créer bucket S3
scw object bucket create \
  name=mecapy-storage \
  region=fr-par
```

Semaine 2 : API + Worker

```
 # 1. Déployer API sur Clever Cloud
clever create --type python mecapy-api
clever deploy

# 2. Build worker image
docker build -t mecapy-worker:v1 ./worker
docker push registry.scaleway.com/mecapy/worker:v1

# 3. Déployer Serverless Container
scw container container deploy \
  --name mecapy-worker \
  --registry-image mecapy/worker:v1
```

Semaine 3 : Tests + Production

```
 # Test end-to-end
curl -X POST https://api.mecapy.com/tasks/execute \
  -H "Authorization: Bearer $TOKEN" \
  -d '{"code": "...", "inputs": {...}}'

# Monitoring
scw container container logs mecapy-worker
```

🎯 WORKFLOW UTILISATEUR SIMPLIFIÉ

1. Créer une tâche

```
 # SDK Python
from mecapy import MecaPy

client = MecaPy(token="xxx")

task = client.tasks.create(
    name="Calcul contrainte vis",
    code="""
def calculate(inputs):
    import numpy as np
    force = inputs['force']
    section = inputs['section']
    return {'stress': force / section}
""",
    inputs={"force": 1000, "section": 10}
)
```

2. Exécuter

```
 result = task.execute()  # Bloquant
# OU
task.execute_async()     # Non-bloquant
```

3. Récupérer résultats

```
 if task.status == "completed":
    print(task.result)  # {'stress': 100}
```

🔒 SÉCURITÉ (Serverless)

Isolation

```
 # worker/sandbox.py
import RestrictedPython
from RestrictedPython import safe_builtins

def execute_user_code(code: str, inputs: dict) -> dict:
    """Exécute le code utilisateur en sandbox"""

    # 1. Compiler code en mode restreint
    byte_code = RestrictedPython.compile_restricted(
        code,
        filename='<user_code>',
        mode='exec'
    )

    # 2. Whitelist imports
    safe_globals = {
        '__builtins__': safe_builtins,
        'numpy': __import__('numpy'),
        'scipy': __import__('scipy'),
        'pandas': __import__('pandas'),
    }

    # 3. Exécuter avec timeout
    exec(byte_code, safe_globals)

    # 4. Appeler fonction calculate()
    return safe_globals['calculate'](inputs)
```

📊 COMPARAISON FINALE

Critère	Serverless Containers	K8s + Celery	Serverless Functions
Gestion infra	ϟ Zéro	🟡 Élevée	ϟ Zéro
Complexité	✅ Simple	🟡 Complexe	✅ Simple
Limite nombre	✔ Illimité	✔ Illimité	✖ 1000
Timeout	⚠ 10min	✔ Illimité	⚠ 15min
Coût base	€0	€50/mois	€0
Coût scale	€€	€€	€€€

Critère	Serverless Containers	K8s + Celery	Serverless Functions
Flexibilité	✅ Docker	✅ Total	❌ ZIP only
Recommandé solo	✅✅✅	❌	⚠️

✅ RECOMMANDATION FINALE

🎯 Pour démarrer (0-6 mois)

→ **100% Serverless Containers Scaleway** - Coût : ~€50/mois - Maintenance : ~1h/mois - Focus : Développement produit

🚀 Si besoin calculs > 10min

→ **Ajouter instances on-demand** (Scaleway GPU/CPU) - Lancées automatiquement si `estimated_duration > 10min` - Détruites après exécution - Coût additionnel : ~€20-50/mois

📈 Si > 50 000 calculs/jour

→ **Migrer vers K8s + Celery** - Seulement si rentable économiquement - Probablement dans 1-2 ans

---

✅ **Verdict : Serverless Containers = Architecture idéale pour développeur solo**

Document généré le : 2025-09-30 **Version** : 2.0 - Architecture Serverless