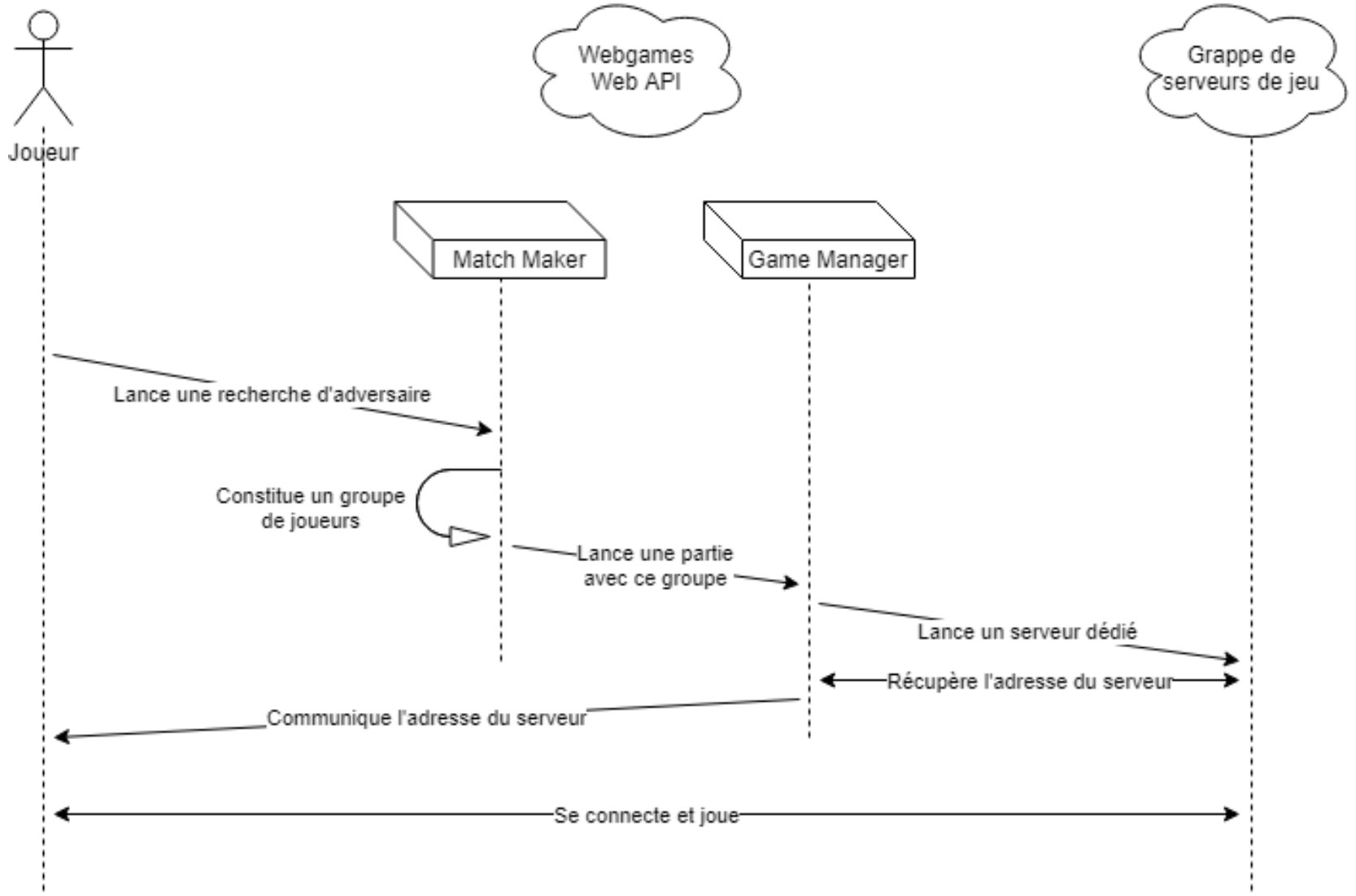
Webgames

Développement d'une application évolutive pour héberger des parties de jeux multijoueurs à la demande

Le principe


Joueur
Webgames
Web API
Grappe de
serveurs de jeu
Match Maker
Game Manager
Lance une recherche d'adversaire
Constitue un groupe
de joueurs
Lance une partie
avec ce groupe
Lance un serveur dédié
Récupère l'adresse du serveur
Communique l'adresse du serveur
Se connecte et joue

Pour qui ?

- Les développeurs de jeux-vidéos
- Les joueurs

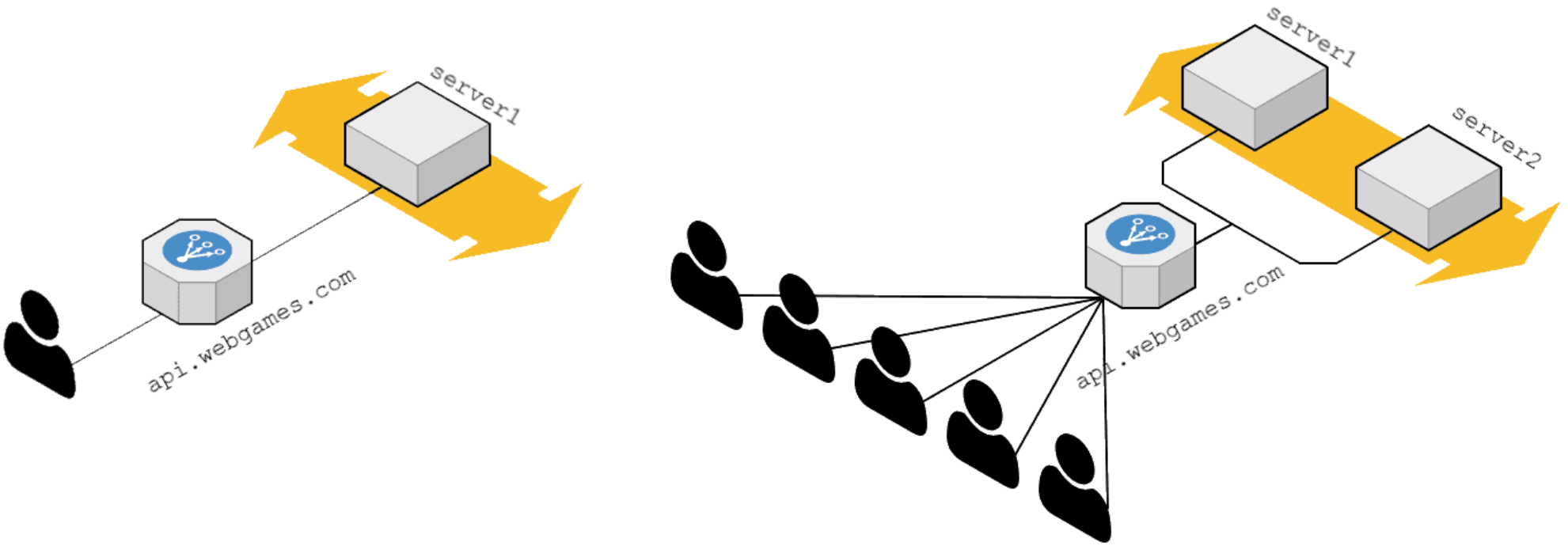
Pourquoi ?

- Aucune solution open-source ou adaptée aux développeurs indépendant.
- Marché potentiellement très lucratif
- Technologies intéressantes

Application Évolutive ?

- Mise à échelle horizontale
  - Containers
  - Orchestrateur
  - Cloud
- Architecture à micro-services
  - Application sans état
  - Modèle réseau asynchrone
- Intégration continue

Mise à échelle horizontale
Horizontale Scaling



Docker : Container

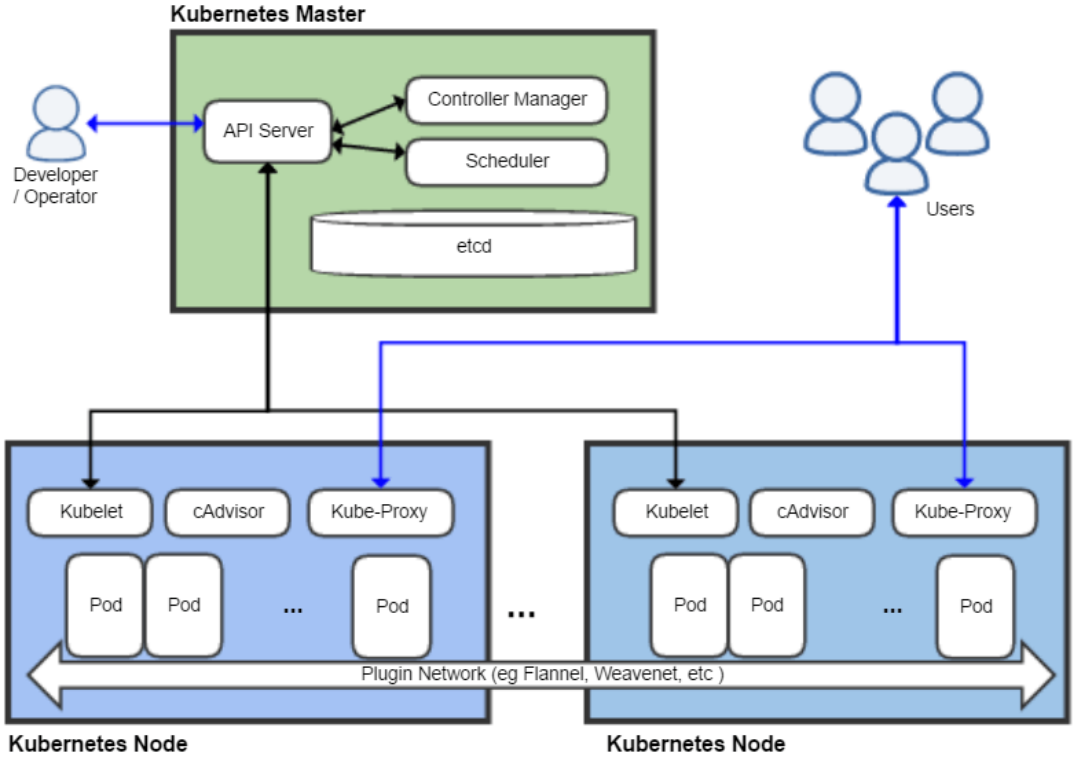
- Crée une image avec l'application et toutes ses dépendances (bibliothèques, logiciels).
- L'image peut être distribuée sur n'importe quel serveur Linux

```
FROM python:3.6

RUN apt install libzmq3-dev
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app
COPY . /usr/src/app
RUN pip install -r requirements.txt

EXPOSE 22548

CMD ["python", "-m", "webapi", "run"]
```

```
$ docker build . --tag webapi

$ docker run webapi --port 22548
$ curl localhost:22548/status
Server is running !
```

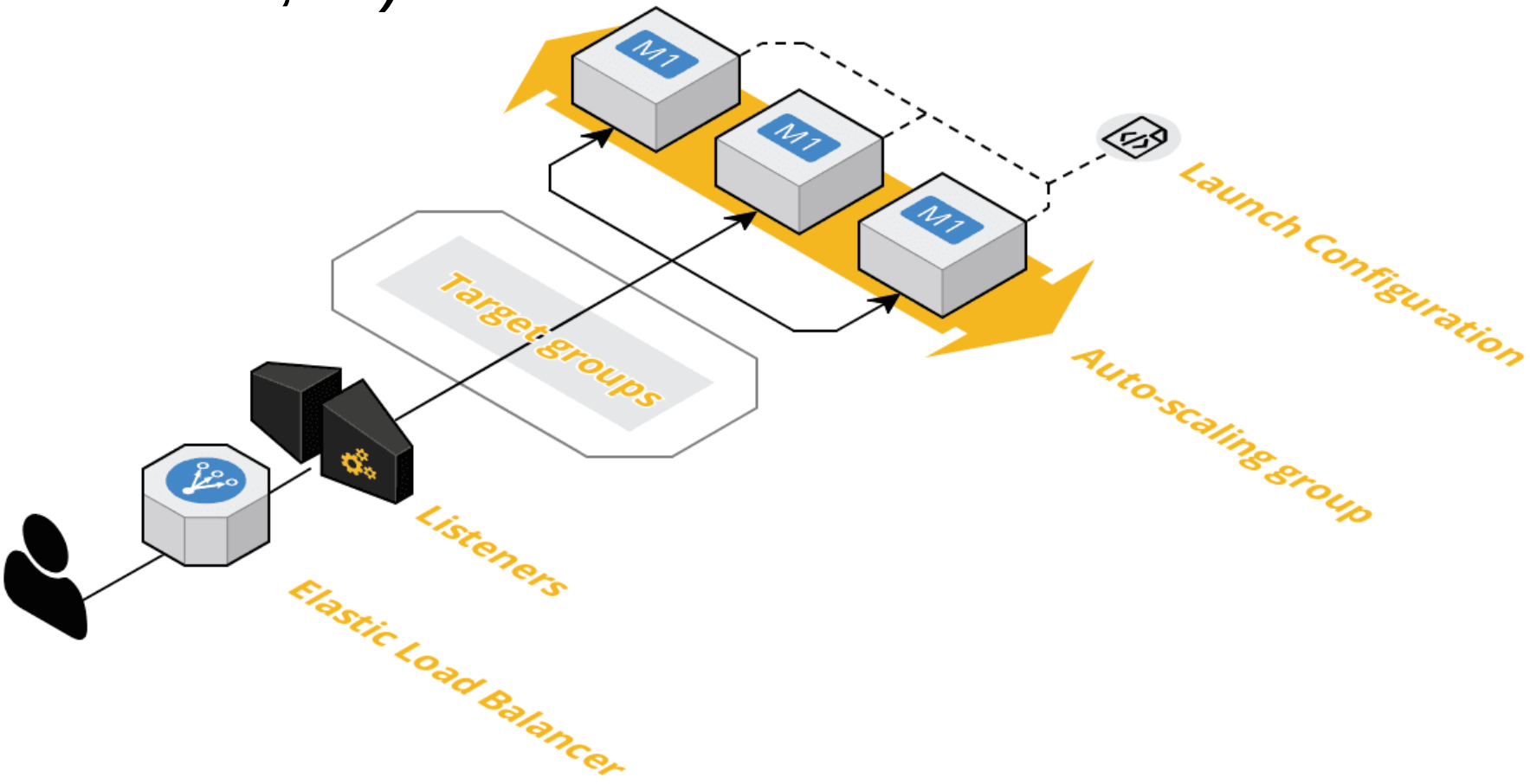
Kubernetes : Orchestrateur

- Répartition des containers sur l'ensemble des serveurs physiques
- Lancement/arrêt de containers en fonction de la charge utilisateur (requêtes par seconde, délai)



Amazon Web Services : Cloud

- Lancement/arrêt de serveurs *physiques* en fonction de la charge des serveurs (CPU, RAM, Réseau, ...)



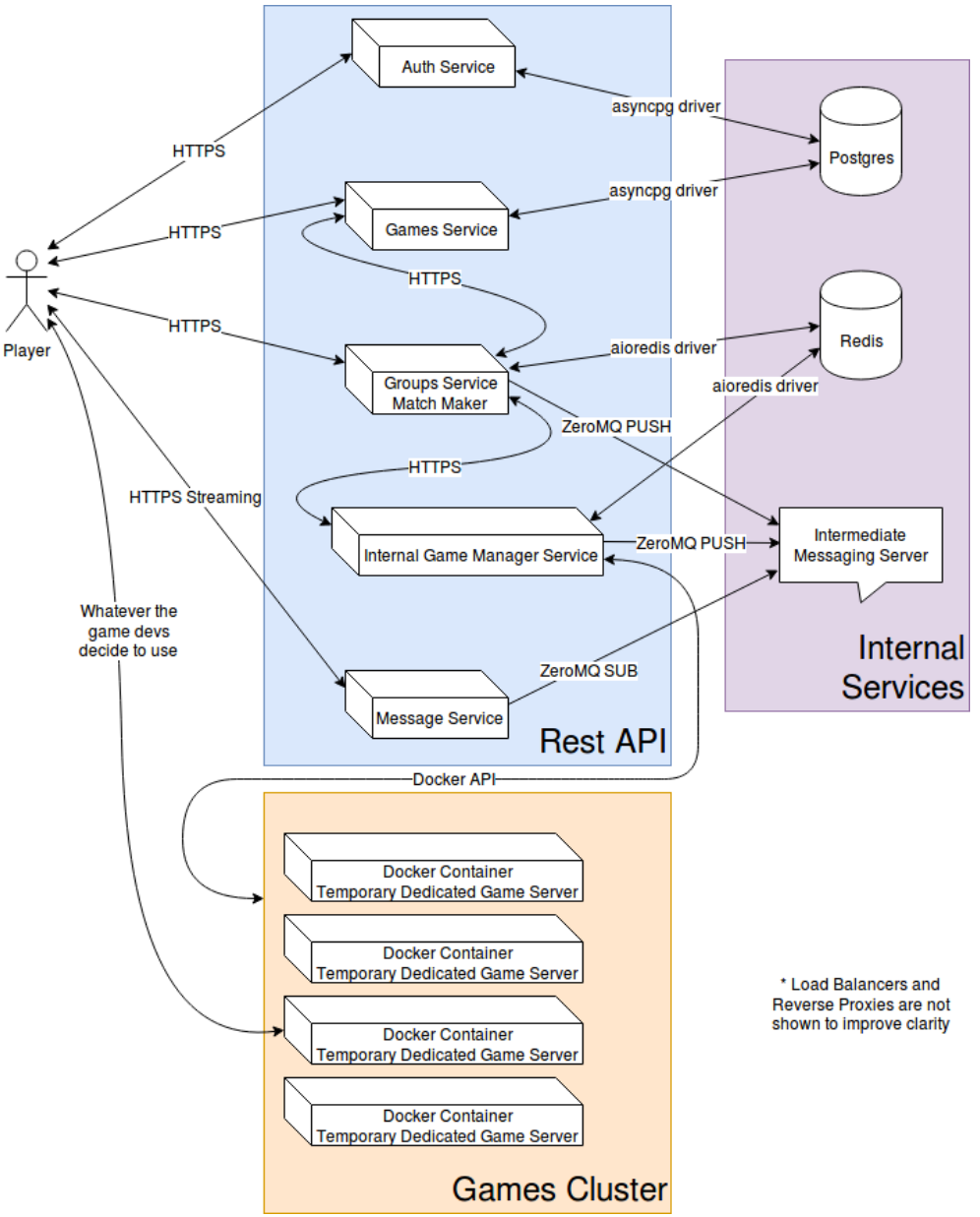
Amazon Web Services : Cloud

- Lancement/arrêt de serveurs *physiques* en fonction de la charge des serveurs (CPU, RAM, Réseau, ...)

120€ / mois

Architecture à Micro-Services

- Chaque service fait une seule chose et le fait bien
- Plus résilient aux crashs
- Mises à jours simplifiées
- Mise à échelle ciblée



Application sans-état

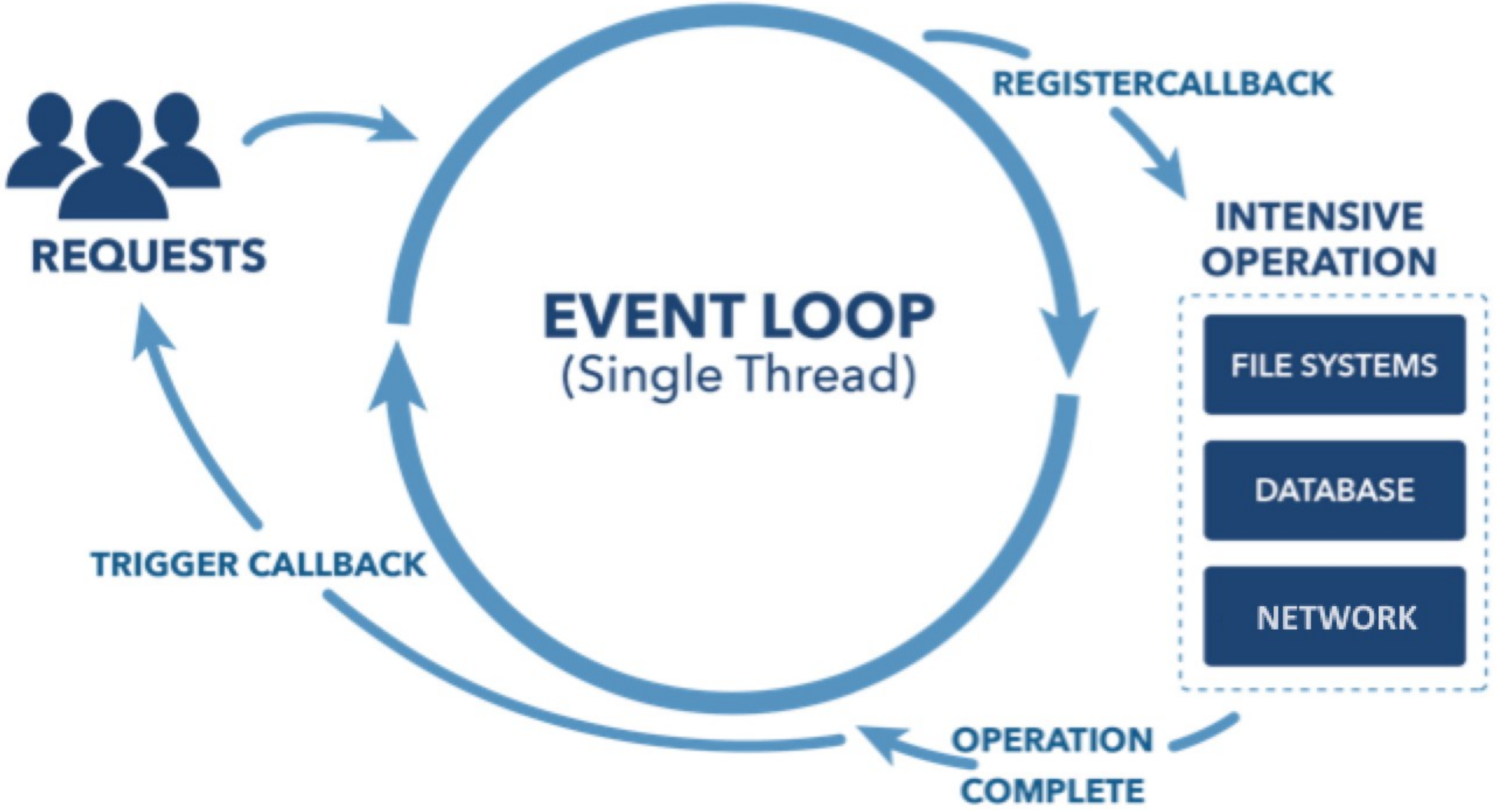
- Toutes les requêtes se suffisent à elles-mêmes
- Nécessaire pour une mise à échelle horizontale

- ➔ API Rest
- ➔ JSON Web Tokens
- ➔ Redis

Modèle réseau asynchrone

- Modèle réseau non-bloquant.
- Permet de gérer plusieurs milliers de connexions parallèles sur un même thread.
- Fonctionne sur le principe d'une boucle événementielle.
- Nécessite que **tout** le code soit asynchrone.
- « The performance of uvloop-based asyncio is close to that of Go programs. » Yury Selivanov, Python code developer

Modèle réseau asynchrone boucle événementielle



Intégration continue

- Contrôle de version 

- Tests unitaires et d'intégration
- Vérification des dépendances 

- Statistiques
  - Qualité du code 

  - Couverture des tests 

- Création du package python 

- Création de l'image docker

Hébergement à la demande ?

- Constitue des groupes de joueurs
- Lance des serveurs de jeu dans des containers dédiés à ces groupes de joueurs
- Arrête les containers une fois la partie finie

- ✔ Aucun serveur de jeu superflu

Démo !