Phone Tycoon

Documentation technique

Theo DESHAYES
Richard LAROZE
Krysztof DUDEK
Valere TETELIN

1 - Présentation générale de l'architecture et des motivations

L'application "Phone tycoon" est un jeu qui doit sensibiliser les joueurs des impacts environnementaux suite à ses choix. Au début de la partie, le joueur n'est pas confronté aux impacts environnementaux, le but est d'appuyer que des choix anodins peuvent avoir des impacts insoupçonnés.

Vous pouvez jouer au jeu a cette adresse http://pfa-dev.hetzner.infra.tetel.in:81/ avec la derniere version duj 05 mai 2020.

Le jeu a été conçu dans le but de fonctionner sur un ou plusieurs clients physiques, avec un écran tactile et à disposition du public de l'exposition.
Afin de controler le deroulement du jeu, effectuer des modifications, et sauvegarder les différentes statistiques des joueurs en plus de ce client, le projet contient un serveur "maître" qui contrôle les clients depuis une base centrale.

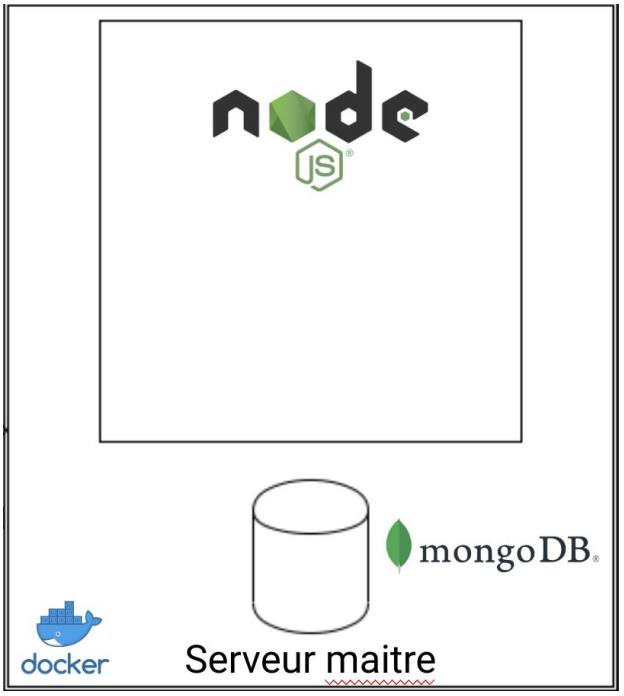
Suite aux événements, l'architecture a légèrement été adaptée afin de pouvoir être joué sur le web.

Outre cette configuration de production, il est possible de lancer toute l'architecture de manière locale simulant une architecture déployée, c'est cette méthode que nous utilisons afin de développer et que nous recommandons pour installer le jeu à des fins de démonstration.

1-1/ Le serveur maître

Le serveur maître est le centre du jeu, c'est le référentiel pour les questions du jeu, les statistiques, etc … Le client en se connectant va récupérer ces informations depuis ce service et ensuite initialiser son jeu.

Voici la composition du serveur maître:



Ce serveur est déployé sur une seule machine et embarque un serveur MongoDB, MongoDB est idéal pour notre cas d'utilisation car il permets de stocker des documents de manières rapides et sans relations.

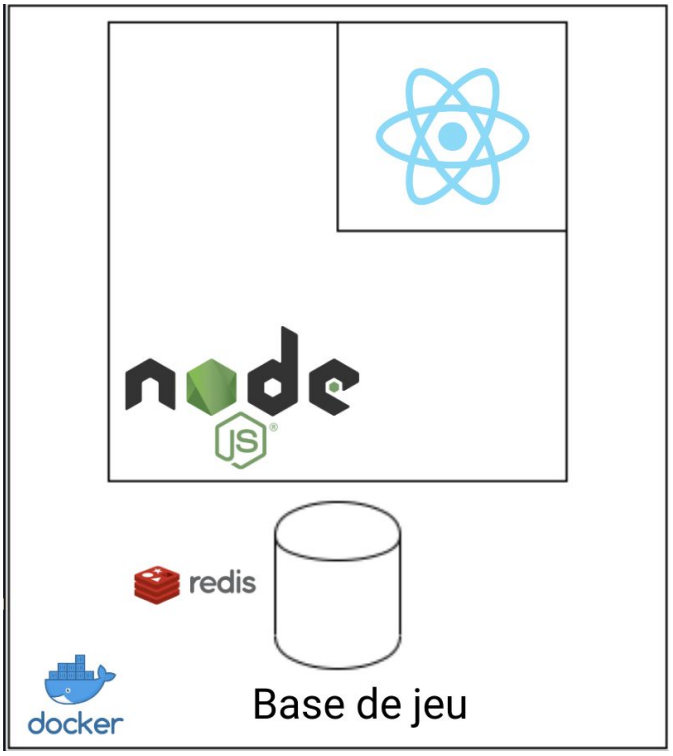
Ce service a aussi pour tâche de sauvegarder les statistiques des joueurs. Il les expose par partie et trie par date de fin de partie. Ce service contient également les statiques de performances, d'impact environnemental et les faits sur les lieux d'extractions des composants choisis.

Dans le mode standalone, il est aussi accompagné d'un processus "one-shot" data-importer qui vise a alimenter la base de données dans ce mode a partir d'un fichier JSON.

1-2/ Le client de jeu

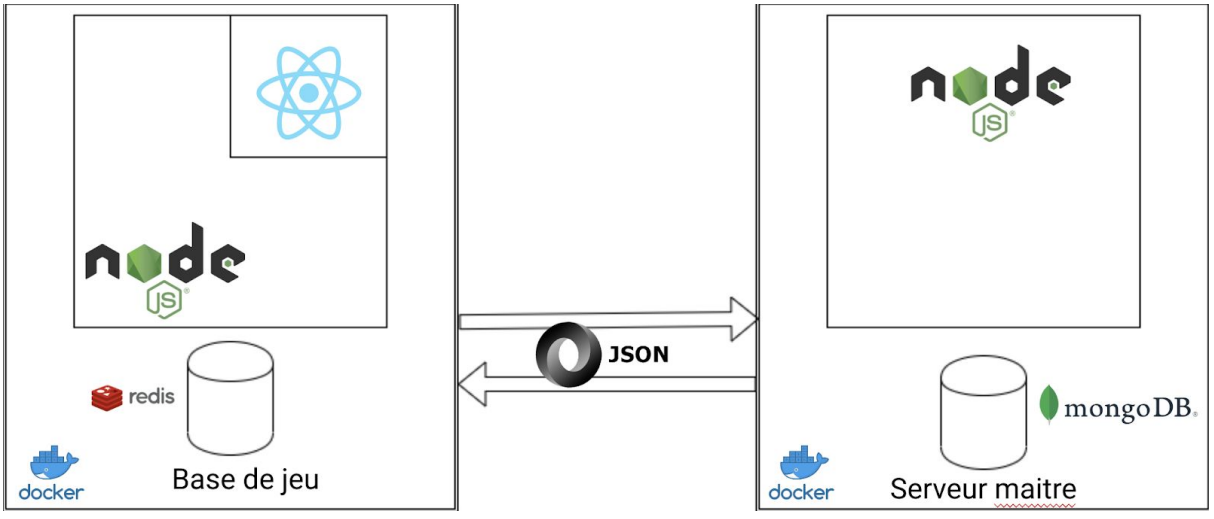
Le client de jeu est déployé sur une machine physique ou alors sur un serveur web. Dans le cas ou la machine physique pourrait être coupée d'internet, certaines données sont mises en cache dans un redis embarqué, afin de pouvoir continuer le jeu malgré une coupure brève. Une coupure de longue durée peut néanmoins entraîner une dégradation progressive de l'expérience de jeu.

Schema du client du jeu:



Le client est composé d'un middleware tout d'abord qui sert de serveur web et est écrit en NodeJS, il communique avec un cache redis afin de stocker les données dont il pourrait avoir besoin en mode hors ligne et sert de relai entre les commandes venant du navigateur et le serveur maître pour les opérations en ligne (envoi/récupération de statistiques de parties).
Il expose également une version de production de l'interface qui est faite en ReactJS avec l'aide de Redux, Bulma.

1-3/ Schema general d'architecture



2 - Installation de l'application en "standalone"

Pré-requis:

- Docker
- Docker-Compose

2-1/ Quick-Start

La procédure d'installation la plus rapide et la plus simple est la méthode "quickstart", elle consiste à récupérer des images précompilées de tous les composants de l'application et assure un fonctionnement de l'application si Docker et Docker-Compose sont fonctionnels sur votre machine.

Une fois que vous avez clone le projet vous pouvez exécuter cette commande:

```
docker-compose -f quickstart.yml up
```

Docker va ensuite télécharger les images de notre application ainsi que les images de base de donnée. Il va ensuite les lancer avec les paramètres que nous avons préconfigurés.

Afin de mettre a jour il vous suffit de lancer cette commande:

```
docker-compose -f quickstart.yml pull
```

Puis de relancer, afin de nettoyer le cache a chaque mise a jour faites un:
DELETE http://localhost:81/api/components

2-2/ Compilation des sources via Docker

Si vous avez effectué des modifications personnelles au projet et que vous souhaitez lancer cette version, vous pouvez compiler le projet a partir des sources tout en profitant de Docker et de Docker Compose afin de paramétrer automatiquement l'infrastructure.

Pour cela, au lieu de construire le fichier "quickstart" vous allez tout simplement construire le projet:

```
docker-compose up --build
```

Cette opération est longue, surtout la première fois car Docker va re-compiler un a un tous les composants de l'application.

Après quelques minutes si tout s'est bien passé vous devriez pouvoir accéder à l'interface via http://localhost:81/

4 - Installation du serveur seul

Ceci est la méthode de déploiement en production recommandée, en revanche elle est plus complexe a mettre en oeuvre et nécessite un certain temps de configuration.

Pre-requis:

- Docker
- MongoDB

Après avoir clone le repo, vous devez construire l'image Docker a partir des sources. Pour ce faire entrez dans le répertoire serveur et lancez

```
docker build -t cotc-server .
```

Après quelques secondes l'image doit être construite. Vous pouvez désormais utiliser docker run afin de lancer le serveur. Afin de configurer votre serveur MongoDB vous devrez préciser les variables d'environnement suivantes:

- DB_HOST -- L'adresse du serveur de BDD
- PORT -- Le port ou l'api est exposé

Par défaut, la connexion a la base de donnée n'est pas authentifié, dans cette configuration il faut bien veiller a ce que la base de données soit locale a l'application et qu'elle ne soit pas exposée sur internet. Si vous avez besoin de mettre votre base de données sur une autre machine ou que vous souhaitez y accéder via Internet, merci d'éditer le fichier server/models/index.js afin d'ajouter le support de la connexion authentifiée.

Vous pouvez ensuite lancer l'application avec:

```
docker run cotc-server <... vos parametres>
```

3 - Installation du client seul

Pré-requis:

- Docker
- Redis

Le client est la dernière pièce de l'application, étant donné que le serveur Redis n'est utilisé que pour mettre en cache certaines données, il peut être volatile et avoir une durée de vie restreinte.

Vous devez donc construire la aussi le client a partir des sources, pour ce faire entrez dans le répertoire client puis exécutez cette commande:

```
docker build -t coctc-client .
```

Après quelques minutes l'image est finalisée, vous devez préciser ces variables d'environnement afin de connecter votre client au serveur master et au redis:

- REDIS_HOST -- Hostname du serveur redis
- SERVICE_HOST -- Adresse du serveur maître
- SERVICE_PORT -- Port du serveur maître

Vous pouvez ensuite lancer le client avec:

```
docker run cotc-client <... vos paramètres>
```