SAMAGAMES

CAHIER DES CHARGES

HYDROANGEAS

RÉALISATION

NOPOZA

TYPE

INFRASTRUCTURE

CONCEPT

GESTIONNAIRE DE RESSOURCES PERMETTANT L'ADAPTATION À LA DEMANDE

RÉALISÉ EN NOVEMBRE 2015

DESCRIPTIF RAPIDE DU PROJET

L'Hydroangeas est une application standalone client/serveur qui à pour rôle de gérer toute les ressources de l'infrastructure afin de l'adapter à la demande.
C'est une application centralisé qui peut se lancer en mode serveur ou en mode client.

Vocabulaire :

Hydroangeas	Serveur ou Client centralisé (Diminutif "Hydro")
Serveur de jeu	Instance de jeu de Minecraft
HUB	Instance de Minecraft Lobby/HUB servant de répartiteur
Machine	Machine dédié sous debian wheezy
Template	Version (Snapshot) unique à un instant T d'un type de jeu
Coupaing	Youtubeur / Streamer reconnu et gradé sur les serveurs de jeu
BungeeCord	Reverse Proxy Minecraft. Point d'entrer de l'infrastructure

- Connaissance de toutes les arènes disponibles en temps réel avec leurs informations serveur.
- Connaissance de tous les serveurs disponible avec leur capacités
- Répartition des serveurs en fonction de la charge de chaque jeux.
- Encodage des informations des panneaux pour envoi direct au lobby.
- Mise à jours automatique des jeux grâce à un système de 'template' unique
- Communications Pub/Sub Redis.
- Console centrale en bash pour accès global et portabilité
- Gestion des templates Bukkit/Plugins/Config/Maps avec versionning.
- Le serveur stockera en cache ses données sur un serveur redis local afin d'éviter un crash total en cas de reboot.
- Chaque serveur stockera des actions prédéfinies en cas de non réponses du Hydroangeas.

Fonctionnement :

Serveur:
Application “Hydro” lancée en mode “serveur”
2 Parties, la première qui est le “cerveau” s’occupe de lire les données actuelles et de réfléchir aux actions à faire.

La deuxième partie s’occupe de la communication, lorsqu’un serveur envoi une demande à
l’Hydroangeas ce processus s’occupe de gérer sa demande ainsi que de lui répondre ou de stocker l’information en fonction de l’évènement.

Client :
Application “Hydro” lancée en mode “client”
Agent personnalisée qui s’occupe d’établir une connexion avec l’Hydroangeas afin d’envoyer des données et de recevoir ses ordres. Il remonte les informations en temps réel.

Il gère la création et la suppression des serveurs, ainsi que la mise en cache des données et la sauvegarde de certaines informations.
Le client communique avec la RestFull API afin d’interagir avec la base de donnée SQL.

Arborescence de la racine client:

```
Hydroangeas
├── config.json
├── Hydroangeas.jar
├── Hydroangeas.log
├── run.sh
├── servers
│   ├── cache
│   │   ├── dependencies
│   │   ├── maps
│   │   └── servers
│   ├── Hub_X
│   │   ├── logs
│   │   │   └── latest.log
│   │   ├── plugins
│   │   ├── server.properties
│   │   ├── spigot.jar
│   │   ├── spigot.yml
│   │   ├── usercache.json
│   │   ├── whitelist.json
│   │   └── world
├── start.sh
├── stop.sh
└── update.sh
```

Nous n'avons fait apparaître que les dossiers et fichiers les plus pertinents.

config.json
```
{
 /** Common **/
 «redis-ip»: «XXXX»,
 «redis-port»: XXXX,
 «redis-password»: «XXXX»,
 «web-domain»: «XXXX»,
 «restfull-url»: «XXXX»,
 «restfull-user»: «XXXX»,
 «restfull-password»: «XXXX»,

 /** Client **/
 «max-weight»: XXXX
}
```

UML :
http://bit.ly/2u8S2rf

Partie technique:

Partie Serveur:
Il n'existe qu'un seul et unique HydroServer et il communique exclusivement par le biais de Redis Pub/Sub
En cas de redémarrage de l'HydroServer, il récupérera toute les information automatiquement à son démarrage.

Lors d'un démarrage, il commence tout d'abords par se connecter à Redis. Il s'inscrit a un channel dans le pub/sub de redis o il recevra toutes ses données sérialisées en JSON. Les données sont désérialisées en fonction du protocole qui est contenu dans le package "common/protocol", cette méthode est fortement inspirée de Netty mais est appliquer sur du pub/sub pour des soucis pratiques de connexion (pas de gestion de port ni d'ip ou d'authentification juste une connexion a redis)

Ensuite le serveur envoi un packet dans le channel global (C'est à dire sur tous les channel) afin d'informer tous les clients déjà présents d'envoyer leurs informations au serveur afin que celui-ci se mette à jours. L'HydroServer patientera 40 seconde, le temps que toutes les informations soient reçus (c'est 2 fois le temps d'un KeepAlive).
Les templates sont chargées au démarrage, puis les queues sont crées. Actuellement chaque template dispose de sa propre file d'attente même si certaines sont inutiles (au hasard les file d'attentes pour les Hub).

Ensuite le hub balancer est lance au bout de 40 secondes et enfin au bout de 5 min le system de cleanup des serveurs (son fonctionnement sera detaille plus loin).

- Les templates

Il y a 2 types de templates:
Le premier qui est dit “simple” est appeler par son ID et contient :

Information	Clé	Nature
Nom du jeu		string
Nom de la map		string
Le poids du “template”		int
Nombre maximum de slot avant de démarrer		int
Option de jeu		JSON (string)
Options JVM		JSON (string)
Si elle a été commandé par un Coupaing		boolean

Le second type est sous la forme de package, il contient plusieurs templates “simples”, lors de la commande d’un serveur un des template est choisit au hasard et est envoyer au client. Ce second template est surtout utilise pour faire varier les maps dans les jeux d’arcade, le choix est donc au hasard (obviously) et non pas par le joueur, ce qui se traduit par un seul panneau.

- Les files d’attentes
Il existe une fil d’attente par type de jeu (Template)
Une file d’attente se présente sous la forme d’une PriorityQueue. Pour des raisons de simplicité, chaque place de la queue contient un groupe de joueurs de 1 à X joueurs.

Le groupe est composé de l’uuid de son leader, de la liste des joueurs dans le groupe (leader compris et sous la forme d’uuid) aussi que de la prioritee du groupe.

La priorité va de 0 a + l'infinie en sachant que 0 est la meilleure priorité.
C'est le HUB qui indique à HydroClient les UUID et la Priorité du groupe car HydroServer n'a pas à connaître les grades de chaque joueurs.
La prioritee du groupe est celle du joueur qui a la prioritee la plus importante. Donc par exemple si un groupe de joueurs lambda ajoute un Administrateur dans leur groupe alors le groupe se verra attribuer la priorité de l'administrateur.

L'ajout d'un groupe dans la queue se fait d'abord par sa creation avec uniquement son leader et ensuite, si il y en a, les autres membres du groupe.
A chaque modification d'un groupe (ajout/suppression d'un joueur) le groupe est enlever de la queue puis traiter (modification des joueur et calcul de la nouvelle prioritee de groupe) et enfin reajouter a la queue. La queue s'occupe de trier en fonction de la prioritee afin que ceux qui ont une prioritee plus importantes passent devant les autres.
Ainsi l'ajout se fait sur le Hub qui envoie les bon packets avec une implementations partielle du protocol de son coter, et bungeebridge s'occupe d'envoyer les packets de suppression lors de la deconnexion.

Ensuite une fois que la queue est remplie: comme dit precedement a chaque queue correspond un template. Aussi differents thread sont present dans la queue.
Le premier qui sert uniquement a envoyer les infos aux hub via pub/sub pour afficher sur les panneaux.

Le second sert à informer le joueur qui est present dans la queue, sa position dans la queue, combien de personne sont necessaires pour le demarrage d'un serveur ou si le serveur est demarrer et les personnes qui composent le groupe.
Enfin le troisieme et plus interraissant est celui qui gere le fonctionnement de la queue:
Celui si va recuperer a intervalle constant les serveurs en train de demarrer ou qui attendent des joueurs (le status du jeu est gerer dans l'api de jeu et partiellement implementer dans hydro), ensuite on recupere le nombre de joueurs dans la queue qui correspond au nombre de slots dispo sur le jeu et on les envoi sur le serveur (a l'aide d'un packet pub/sub implementer dans l'api de jeu). Enfin il va verifier si il est necessaire de demarrer un serveur. Si la liste precedante de serveur est vide ou que tous les serveurs demarrer sont pleins alors ont demandera l'ajout d'un serveur mais un serveur sera rellement demarrer uniquement si la queue contient au moins le nombre de joueurs necessaires pour demarrer la partie (min slot dans le template) ou si un override est demander.
L'override se base sur des stats. Si plus d'un serveur demarre en 2 minutes alors la queue passe en mode anticipation et donc un serveur démarrera forcement si il n'y en a aucun de dispo. Ces stats se font sur 10 min.

Le thread d'info des joueurs verifie à chaque fois si le troisiemme thread est toujours en fonctionnement et si il a crash pour une raison ou une autre, ce thread relance le troisieme thread avec les erreures adequates dans la console.

- Clean Server (Security shutdown)

Le CleanServer (ou SecurityShutdown) est une mesure de vérification servant de "garde-fou" qui vérifiera régulièrement si le "temps de vie" d'un serveur dépasse son "temps de vie maximum". L'intervalle de vérification est actuellement de 5 minutes.
Lors du démarrage d'un serveur de jeu, sa durée de vie maximum est de 2min30sec. A partir du moment où il commence à recevoir des joueurs, sa "durée de vie maximum" est augmenté à 4 heures.

La queue envoie des joueurs quand l'api sur le serveur a indiquer un status joignable c'est donc grace a ca que l'on sait que le demarrage s'est deroule correctement.

- Hub Balancer

Le HubBalancer s'occupe de répartir efficacement et de façon homogène les joueurs sur les différents HUB disponibles et de démarrer/arrêter de nouveau HUB en fonction du nombre de joueurs déjà présents.

HydroServer gère tous les HydroClient, mais il stock également les informations de chaque HUB dans Redis.
BungeeCord, à partir du moment où il reçoit la demande de connexion d'un joueur, va interroger les données présentes sur Redis et choisir sur quel HUB il va envoyer le joueur.
HydroServer ne fait que gérer les machines et indiquer la charge de chaque HUB, c'est ensuite a BungeeCord de choisir ou il doit envoyer ses joueurs.
Cette communication se fait par un stockage ordonnée dans redis.
Ensuite le HubBalancer contient un Thread de verification.

Il calcule le nombre idéal de HUB, ce nombre est calculé par le nombre de joueurs actuels sur les HUB multiplié par 1.6 (Magic Value), divisé par le nombre de Slots maximum que peut accueillir un HUB, puis on ajoute une marge de sécurité de 2 HUB (ce qui correspond a la charge à vide) on prend ensuite la valeur entière supérieur ce qui nous donne notre nombre de hub idéal.
Formule : [(NbJoueursHub * 1,6) / MaxSlotHub] + 2

Si cette valeur indique qu'il y a actuellement trop de HUB lancée, Hydro-Server va tenter de fermer les HUB les plus anciens ET les moins peuplés. Il "évacue" (C'est à dire qu'il renvois les joueurs sur d'autres HUB), puis "éteints" (C'est à dire qu'il stop le serveur MC) et enfin "désinscrit" (c'est à dire qu'il le retire de la liste des HUB inscrite dans Redis pouvant acceuillir des joueurs par BungeeCord), le HUB.

Creation de serveur
TODO

Commandes:

/Commandes	Informations
order	Liste des Templates disponibles
order <TemplateName>	Commande un serveur
info	Affiche tous les clients
info <ClientID>	Affiche les détailles de l'Hydroclient
refresh	Demande à tous les clients d'envoyer leurs informations
reload	recharge les templates
shutdown <server / client> <ClientID / ServerName>	Demande l'arrêt de la cible
stop	éteint l'Hydro sur lequel vous êtes

- Partie client
TODO