Architecture micro-service

TP 3 - Serveur TCP/IP simple - c. Limites

Philippe Roussille

3iL INGENIEURS
</les sciences informatiques>

1 Limites de l'architecture actuelle

Le serveur IRC fonctionne. Il tient en mémoire les utilisateurs, les canaux, et dialogue en TCP. Mais que se passe-t-il **si on change d'échelle**, de protocole, ou de contraintes ?

2 Des questions pour cibler votre réflexion

2.1 Couplage et contrôle des responsabilités

- **Quelles fonctions du serveur manipulent à la fois l'état métier, les sockets réseau et le journal ?** Pouvez-vous identifier un exemple de fonction violant le principe de séparation des responsabilités ?
- **Si vous supprimez `IRCHandler` et branchez un autre protocole (HTTP, WebSocket...), combien de fonctions métier devrez-vous réécrire ?**
- **Le protocole est-il une interface explicite du système ou seulement un comportement émergent ?** Autrement dit : peut-on découpler les « effets » des commandes de leur format textuel ?
- **Quelles sont les opérations atomiques ? Peut-on garantir la cohérence du système si une commande échoue à mi-chemin ?** (Ex : `/msg` réussit à moitié : certains reçoivent, d'autres non)

2.2 Protocole et interopérabilité

- **Quels types d'erreurs le protocole peut-il exprimer ?** Différencie-t-on une erreur de syntaxe, une interdiction d'accès, un état illégal (ex : pas de canal), une erreur serveur ?
- **Pouvez-vous formaliser le protocole actuel (syntaxe, contraintes) sous forme de grammaire normalisée ?**
- **Comment un client non humain saurait-il qu'une ligne reçue est un message utilisateur, une info système, une alerte ?**
- **Ce protocole peut-il être *versionné* ? Peut-on prévoir une rétrocompatibilité ?**

— **Quel est le coût (code, test, fiabilité) de rajouter une commande dans ce protocole par rapport à une API REST bien conçue ?**

2.3 Testabilité et fiabilité

— **Quels tests unitaires sont aujourd'hui impossibles à écrire sans simuler une socket TCP ?** Pourquoi ? Que faudrait-il isoler ?
— **Quels comportements sont aujourd'hui implicites et non testés ?** (Exemples : la disparition silencieuse d'un canal vide, la déconnexion d'un client fantôme, l'ordre d'arrivée des messages)
— **Peut-on simuler un client malveillant qui envoie `/msg` sans `/join` ?** Comment le serveur réagit-il ? Peut-il être amené à un état invalide ?
— **Le système actuel tolère-t-il les pannes partielles ?** (Ex : perte de la socket d'un client, écriture impossible dans le journal, canal corrompu)

2.3.1 Scalabilité et distribution

— **L'état du serveur est-il réplicable ?** Peut-on faire tourner deux instances concurrentes sans conflit ? Pourquoi pas ?
— **Quelles ressources sont globales et quelles données pourraient être distribuées ?** (Ex : la liste des utilisateurs connectés ? les messages d'un canal ?)
— **Que faudrait-il pour brancher un** système de persistance robuste** (base de données ou message queue) à la place du JSON et du `wfile` ?**
— **Le système actuel est-il capable de gérer des charges réseau variables ?** Que se passe-t-il si 1 000 clients se connectent, puis envoient 10 messages chacun dans la même seconde ?
— **Quelle architecture (cluster, bus, micro-services, brokers, etc.) serait capable d'absorber cette charge avec fiabilité ?**

2.3.2 Évolutivité du code et découpage en services

— **Quelles commandes actuelles pourraient être déléguées à un service distinct (externe) ? Pourquoi ?**
— **Peut-on identifier une ou plusieurs interfaces métier dans ce système ?** Que faudrait-il pour que le serveur devienne une simple "coquille réseau" pilotant des services internes ?
— **Que manque-t-il à ce projet pour le rendre *devops-compatible* ?** (Tests, logs structurés, monitoring, API REST, configuration, conteneurisation…)
— **À quelles conditions ce serveur peut-il devenir une base pour une architecture micro-services ?** (Critères : isolation, communication explicite, synchronisation, persistance, monitoring…)

2.4 Travail à rendre de réflexion sur les limites

Rédigez (en binôme) une **synthèse technique argumentée** en vous appuyant sur une sélection des questions ci-dessus. Vous pouvez organiser votre réflexion autour de ces axes :

— Ce que ce serveur fait bien (simplicité, réactivité, etc.)

— Ce qu'il cache ou gère mal (accès concurrents, erreurs silencieuses, non modularité…)
— Ce qu'il faudrait refactorer pour évoluer vers un système web ou micro-service