



AR_elves-rings.jpg

Six solutions, trois vues

Six logiciels, décrits par la manière dont ils fonctionnent. Aucune de ces descriptions ne contient de schéma : c'est à vous de l'en extraire.

Les **trois premiers** se traitent ensemble, au tableau. Les **trois suivants** en groupe.

Consigne

Pour chaque énoncé, trois passes, dans cet ordre.

Passe 1 — Dessiner les trois vues

Esquissez les **trois**, y compris celle qui vous paraît pauvre d'avance. On ne décrète pas qu'une vue est inutile avant de l'avoir tentée : c'est en la dessinant qu'on s'en aperçoit, et une vue pauvre est un résultat, pas un échec.

Chacune avec les symboles de son formalisme tels qu'ils ont été définis en cours.

Quand l'énoncé mentionne un **interdit**, il fait partie du schéma : ce qu'un élément n'a pas le droit de faire se dessine.

Passe 2 — Élaguer

Reprenez chaque schéma et retirez tout ce qui n'aide pas à comprendre. Un élément se garde s'il est **nommé dans l'énoncé** et si son retrait changerait ce que le schéma fait voir. Sinon il part.

- Deux boîtes qui jouent exactement le même rôle dans le mécanisme : une seule suffit — sauf quand c'est leur multiplicité qui est justement le sujet.
- Une boîte dont aucune flèche, entrante ou sortante, ne raconte quelque chose : elle décore.

- Une série de flèches barrées identiques : en garder une ou deux, représentatives, et écrire la règle en note.

Une vue sert à faire comprendre un mécanisme, pas à inventorier des composants. On choisit ce qu'on montre en fonction de ce qu'on veut faire voir.

Passe 3 — Trancher

Classez les trois vues, de la plus éclairante à la moins utile, et dites pour chacune **ce qu'elle apprend sur cette solution**. Pour la dernière, dites aussi ce qu'elle n'apprend pas : c'est là que le classement se justifie.

Un schéma qui tient en une seule boîte n'est pas forcément raté — c'est parfois tout ce que cette vue a à dire de cette solution. Encore fallait-il le dessiner pour pouvoir l'affirmer.

1. BitTorrent

Pour distribuer un gros fichier, BitTorrent ne le fait pas télécharger depuis une machine unique.

- Le fichier est découpé en **morceaux numérotés** de taille fixe. Celui qui le met à disposition publie un petit **fichier descripteur** contenant la liste de ces morceaux, leur empreinte, et l'adresse d'un **traqueur**. Ce descripteur se récupère ailleurs — sur un site web, par courriel, de la main à la main : il ne vient pas du traqueur, puisqu'il faut l'avoir pour savoir où le traqueur se trouve.
 - Le traqueur est un programme classique, qui tourne en permanence sur une machine dédiée. Un participant s'adresse à lui pour obtenir la liste des autres participants au même téléchargement ; en posant la question, il **s'inscrit lui-même** auprès du traqueur, qui enregistre son adresse et l'ajoute à la liste. Le traqueur ne transporte jamais le fichier, et ne stocke aucun morceau.
 - Chaque participant recontacte le traqueur **à intervalle régulier**, fixé par le traqueur, et reçoit une liste à jour : c'est ainsi qu'il découvre les nouveaux arrivants.
 - Dès qu'une connexion s'établit entre deux participants, les deux s'échangent la **liste des morceaux qu'ils possèdent**. Ensuite, à chaque morceau reçu et vérifié, chacun prévient ses voisins : « j'ai le morceau *n* ».
 - Chaque participant réclame à d'autres participants les morceaux qui lui manquent, et **sert en même temps** à ceux qui les lui demandent les morceaux qu'il a déjà reçus. Il n'y a pas de machine spécialisée dans l'envoi : chacun demande et répond.
 - Un participant qui vient d'arriver n'a rien à offrir ; au fur et à mesure qu'il reçoit, il devient utile aux autres. Plus il y a de participants, plus la capacité totale d'envoi augmente.
 - Aucun participant ne fait confiance à un autre : chaque morceau reçu est vérifié par son empreinte avant d'être accepté, et un participant qui envoie n'importe quoi est simplement écarté par ceux qui s'en aperçoivent.
-

2. La chaîne de compilation LLVM

Compiler un programme, c'est le faire traverser une suite d'étapes dont chacune transforme ce que la précédente a produit.

- L'**analyse Rust** lit du code source Rust et en produit une **représentation intermédiaire** : un langage interne, indépendant à la fois du langage d'origine et de la machine visée. L'**analyse Swift** fait la même chose depuis du code source Swift, et produit **la même représentation intermédiaire**.
 - Cette représentation traverse ensuite deux **passes d'optimisation**. Chaque passe lit la représentation, la réécrit, et transmet le résultat à la suivante. On peut en ajouter une, en retirer une, ou changer leur ordre, sans toucher aux autres.
 - Deux **générateurs de code** transforment enfin la représentation optimisée en instructions pour une machine donnée : l'un pour x86, l'autre pour ARM.
 - Rust et Swift réutilisent donc telles quelles les passes d'optimisation et les générateurs de LLVM, sans rien partager de leur analyse.
 - Aucune étape ne sait quelle étape la suit, ni laquelle l'a précédée. Chacune ne connaît que le **format** qu'elle lit et celui qu'elle écrit.
 - Chaque étape traite la totalité du programme avant de passer la main à la suivante.
-

3. Une boutique en ligne livrée d'un bloc

- Le code est découpé en **modules métier** : Catalogue, Commandes, Facturation, Expédition. Chaque module expose une **façade publique**, et tout le reste de son code est invisible depuis les autres modules. Un outil de vérification fait échouer la compilation si quelqu'un passe outre.
 - Chaque module possède son propre **schéma de base de données**. Aucune requête ne fait de jointure entre deux schémas : pour vérifier un prix ou une disponibilité, Commandes ne lit jamais les tables de Catalogue, il s'adresse à sa façade.
 - À l'intérieur d'un module, le code est réparti en trois niveaux. Le **contrôleur** reçoit la requête web, le **service** applique les règles métier, le **dépôt** parle à la base. Le contrôleur n'a pas le droit de s'adresser au dépôt directement.
 - Le service de Commandes doit prévenir le client quand sa commande est confirmée. Il ne connaît ni le courriel ni le SMS : il déclare une **interface** nommée « Notificateur », dans son propre vocabulaire, et l'appelle. Ailleurs, en dehors du code métier, une classe implémente cette interface en parlant à un serveur de courriel. Le branchement entre les deux est fait au démarrage de l'application ; dans les tests, une fausse implémentation prend sa place.
 - Quand une commande est validée, Commandes publie un **événement interne** « commande validée » sur un **bus d'événements interne**. Facturation et Expédition s'y sont abonnés et réagissent chacun de leur côté. Commandes ignore leur existence, et on peut en ajouter un troisième sans toucher à son code.
 - Tout cela est compilé en **un seul artefact**, livré en une seule image, et s'exécute dans **un seul processus**. Pour tenir la charge, on lance plusieurs copies identiques de ce processus derrière un répartiteur, et toutes parlent à la même base de données.
 - Aucune copie ne retient quoi que ce soit entre deux requêtes : le panier est en base, et le navigateur conserve un jeton. N'importe quelle copie peut donc traiter n'importe quelle requête, et on peut en ajouter ou en retirer à chaud.
-

4. VS Code et ses extensions

- Le logiciel installé se réduit à un **noyau** volontairement réduit : édition, fichiers, interface. Il ne sait rien d'aucun langage de programmation en particulier.
 - Tout le reste vient d'**extensions**, installées séparément. Chacune est décrite par un **manifeste** qui déclare ce qu'elle apporte : commandes, entrées de menu, raccourcis, paramètres, thèmes, langages reconnus.
 - Le manifeste déclare aussi des **événements d'activation**. L'extension Python ne démarre que lorsqu'un fichier `.py` est ouvert ; tant que personne n'en ouvre, elle n'est pas chargée.
 - Les extensions tournent dans un **processus séparé** de celui de l'éditeur. Elles ne peuvent agir sur l'interface qu'à travers une interface de programmation fournie par l'éditeur, découpée par domaines : la fenêtre, l'espace de travail, les langages.
 - Une extension ne connaît aucune autre extension. Elle ne peut ni les appeler, ni savoir qu'elles existent : tout ce qu'elle fait passe par l'éditeur.
 - Concrètement, l'extension Python crée une collection de diagnostics, s'**abonne** aux événements qui signalent qu'un document a été modifié, et y dépose des diagnostics — une zone de texte, un message, une gravité. L'éditeur les affiche : il souligne la zone et propose une ampoule.
-

5. Cloudflare devant un site web

- Le propriétaire d'un site fait pointer son nom de domaine vers Cloudflare, et non plus vers sa propre machine. Son **serveur d'origine** reste en place, mais plus aucun visiteur ne s'y adresse directement : le serveur d'origine **refuse toute connexion qui ne provient pas de Cloudflare**.
 - Quand un navigateur demande une page, sa requête arrive au **centre de données le plus proche** géographiquement, parmi les centaines répartis dans le monde.
 - Ce centre regarde s'il détient déjà une copie valide de ce qui est demandé. Si oui, il répond lui-même et le serveur d'origine n'est pas sollicité du tout. Sinon, il transmet la requête à l'origine, renvoie la réponse au navigateur, et en conserve une copie pour les prochains.
 - La durée pendant laquelle une réponse peut être conservée n'est pas décidée par le centre de données : elle est indiquée par le serveur d'origine, dans un **en-tête** de sa réponse.
 - Le navigateur ne sait pas s'il a parlé au cache ou à l'origine, et n'a pas besoin de le savoir : il envoie exactement la même requête dans les deux cas.
 - Chaque requête contient **tout ce qu'il faut** pour être traitée — l'adresse de la ressource demandée et, s'il y a lieu, le jeton d'authentification. Aucun centre de données ne retient quoi que ce soit entre deux requêtes d'un même visiteur, ce qui permet à n'importe lequel de traiter n'importe quelle requête.
 - D'autres intermédiaires peuvent être insérés sur le trajet — filtrage des requêtes malveillantes, limitation du débit, déchiffrement. Ils s'exécutent **dans le centre de données lui-même**, sur le trajet de la requête, avant le cache, et sans qu'aucun n'ait besoin de comprendre ce que fait le site.
-

6. La surveillance de la qualité des flux vidéo chez Prime Video

Prime Video diffuse en direct des matchs et des chaînes de télévision. Des milliers de flux sont diffusés en même temps, et un défaut (une image figée, un son décalé) peut gâcher la diffusion pour tous les spectateurs d'un flux. Un service surveille donc chaque flux en continu et prévient l'équipe d'exploitation dès qu'un défaut apparaît.

- Pour chaque flux surveillé, le service reçoit une copie de ce qui est diffusé, découpée en **segments** d'une seconde.
 - Un **convertisseur** reçoit directement cette copie et transforme chaque segment en données analysables : une série d'**images** et un **extrait sonore**.
 - Deux **détecteurs** examinent ces données, chacun sous un angle différent :
 - le détecteur d'**image figée** compare les images successives. Si elles restent identiques plusieurs secondes de suite, le flux est figé ;
 - le détecteur de **décalage son-image** vérifie que le son et les images d'un même segment sont bien synchronisés.
 - Un **orchestrateur** pilote le travail. Pour chaque segment de chaque flux :
 - il demande au convertisseur de convertir le segment et attend qu'il ait terminé ;
 - il demande ensuite à chaque détecteur d'analyser le segment ; chaque détecteur lui renvoie son verdict ;
 - il garde trace de l'avancement de chaque flux : quel segment est en cours, quelles étapes sont terminées ;
 - si un détecteur signale un défaut, il envoie une **alerte** à l'équipe d'exploitation.
 - Le convertisseur et les détecteurs ne se connaissent pas : aucun n'appelle les autres. Un détecteur ne connaît que le **format** des images et des extraits qu'il lit, et ignore qui les a produits. Seul l'orchestrateur connaît l'ordre des étapes.
 - Chaque composant est **déployé séparément**, sur sa propre infrastructure, et se met à l'échelle indépendamment des autres. Si les analyses sont lentes, on ajoute des instances de détecteurs sans toucher au convertisseur. Les instances de chaque composant sont placées derrière un **répartiteur**.
 - Les images et les extraits ne passent pas directement du convertisseur aux détecteurs. Le convertisseur les **dépose dans un service de stockage**, et chaque détecteur vient les y relire. Chaque passage d'une étape à la suivante est donc une écriture puis une lecture, à travers le réseau.
-