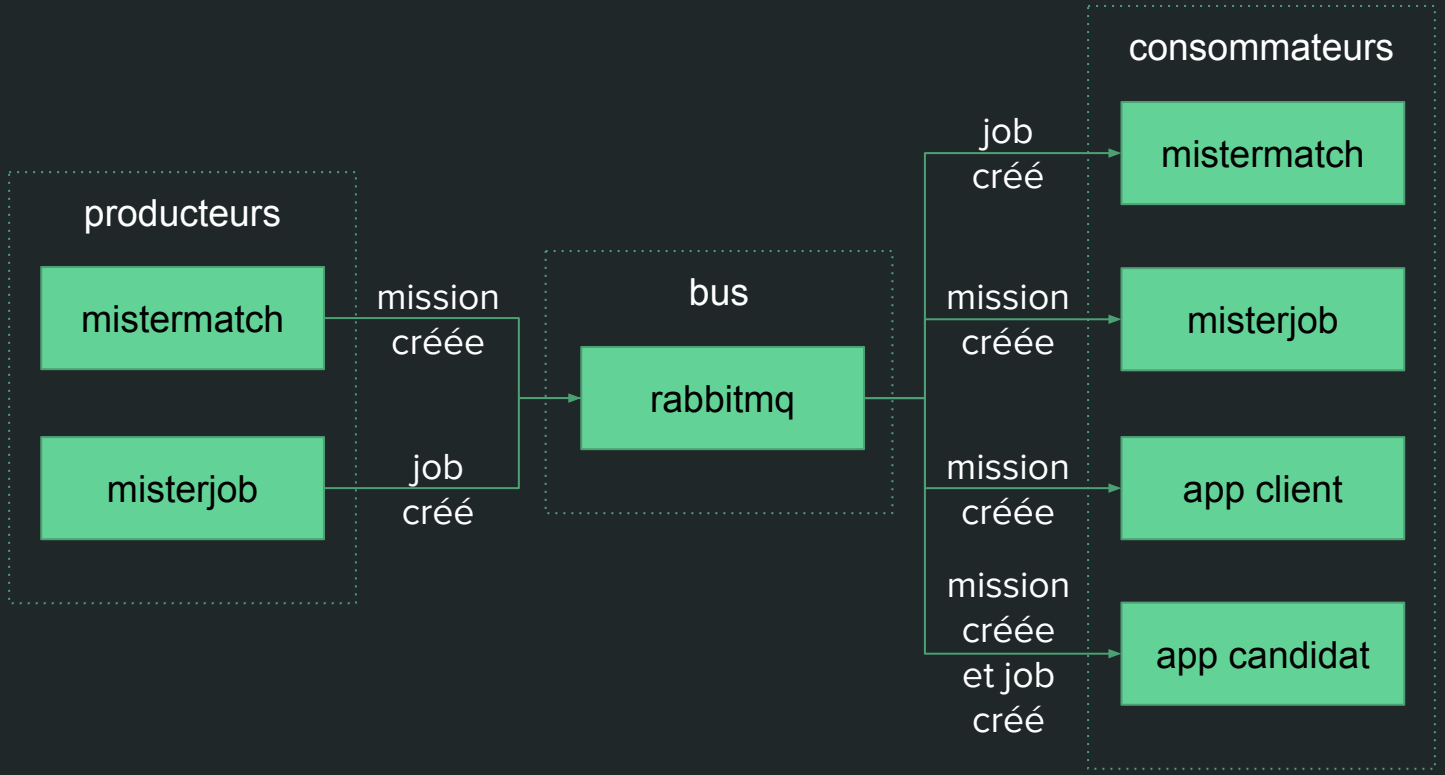
Les architectures orientées événements

Xavier Dutreilh

Définition

- De l’anglais : event-driven architecture (EDA). Parfois, appelées “architecture événementielle” dans certaines entreprises françaises.
- Modèle d’architecture dans lequel des services totalement découplés communiquent entre eux par le biais d’événements.
- 1 événement = 1 changement d’état significatif (ex : création d’une mission dans mistermatch, publication d’une annonce dans misterjob, résultat d’une synchronisation dans tempo, ...).
- Communication asynchrone et couplage lâche entre les services.
- Possibilité de mélanger les langages et les plateformes matérielles/logicielles.
- Fréquemment utilisé dans un contexte de microservices.

Exemple


producteurs
mistermatch
misterjob
mission créée
job créé
bus
rabbitmq
job créé
mission créée
mission créée
mission créée et job créé
consommateurs
mistermatch
misterjob
app client
app candidat

Event Notification

- Pattern dans lequel un producteur émet à destination des consommateurs des événements afin de les informer d'un changement significatif de son domaine.
- Acquittement possible mais indirect des consommateurs sur le traitement réel et effectif des événements.
- Approche minimaliste des événements (ex : ensemble des informations d'une mission suite à sa création dans mistermatch, identifiant d'un établissement suite à sa synchronisation dans tempo, ...).
- Obfuscation de la logique métier dès lors que celle-ci est répartie sur plusieurs événements (ex : synchronisation dans tempo) ⟶ nécessité de monitorer les événements parcourant le système pour avoir une vision claire.

Event-Carried State Transfer

- Pattern dans lequel les services sont parfaitement autonomes au point de ne plus dépendre d’autres sources de données pour fonctionner.
- 1 service = 1 copie locale de toutes les données nécessaires (ex : copie des établissements, missions et viviers de mistermatch dans misterjob, copie des candidats et des établissements de mistermatch dans mistertempo, ...).
- Résilience garantie des services et du système dans sa globalité, surtout en cas de panne des sources de données.
- Diminution des temps de réponse et suppression des goulots d’étranglement.
- Augmentation considérable de la complexité du système.
- Désynchronisation possible des services avec les sources de données.

Event Sourcing

- Pattern dans lequel les changements d'état significatifs des services sont persistés sous forme d'événement dans un store et l'état global des services peut être reconstruit à tout moment en rejouant les événements.
- Approches synchrone (ex : script bin) et asynchrone (ex : misterjob).
- Glissement sémantique : event store = source de vérité et entités métiers + copies locales des sources de données = projections (reconstruites à partir des événements).
- Possibilité d'auditer le système, de revenir à des états spécifiques, voire d'explorer des chemins alternatifs en injectant des événements hypothétiques.
- Intégration difficile avec des services externes (ex : indeed, algolia, ...)

Mistertemp’

- Implémentation hybride mélangeant à la fois une architecture orientée événements, une architecture orientée requêtes et un socle monolithique.
- Utilisation en dilettante du pattern Event Notification (ex : modification des agences et des professions, synchronisation des établissements dans tempo, récupération des contrats et des relevés d’heures depuis tempo).
- Obfuscation de la logique métier entre les services (ex : tempo ↔ mistertempo-v2 ↔ mistertemp-api, misterprofession-v2 ⟶ mistertemp-api, …).
- Utilisation quasi-inexistante du pattern Event-Carried State Transfer.
- Utilisation inexistante du pattern Event Sourcing.
- Pas de monitoring des événements parcourant le système.