Plateforme d'Entreprise Agentique : Spécifications Fonctionnelles Détaillées (Version 1.0 - Projet Pilote : Souscription d'Assurance Automatisée)

Partie I : Vision et Architecture Fondamentale de la Plateforme

1.0 Architecture d'Ensemble : Orchestration de Processus Orientée Événements

1.1 Paradigme Architectural Hybride : L'Orchestration État-Major sur une Chorégraphie Asynchrone

La Plateforme d'Entreprise Agentique (PEA) est fondée sur un paradigme architectural hybride, conçu pour capitaliser sur les forces complémentaires de l'orchestration de processus centralisée et de la chorégraphie événementielle décentralisée. Ce modèle reconnaît la double nature des processus métier modernes : ils exigent à la fois une gouvernance stricte et une agilité d'exécution.

Au cœur de cette architecture, l'orchestration de processus, pilotée par Camunda Platform 8, agit comme un "état-major" stratégique. Elle est responsable de la maintenance de l'état global et de la logique séquentielle des processus métier de bout en bout. Chaque étape, chaque décision et chaque chemin d'exception sont explicitement modélisés et exécutés par le moteur d'orchestration. Cette centralisation offre une visibilité inégalée, une capacité d'audit complète et un contrôle précis sur le déroulement des opérations complexes, ce qui

est indispensable dans un secteur réglementé comme l'assurance.[1]

Cette couche d'orchestration ne communique pas directement avec les services exécutant la logique métier. Elle s'appuie plutôt sur une épine dorsale de chorégraphie asynchrone, mise en œuvre via Apache Kafka. Ce bus d'événements agit comme le "système nerveux central" de la plateforme, découplant l'orchestrateur des services "agentiques" qui réalisent le travail. L'orchestrateur publie des "commandes" sous forme d'événements, et les agents y souscrivent pour exécuter leurs tâches. En retour, les agents publient des "événements de résultat" pour notifier l'orchestrateur de leur progression.[3] Cette approche de chorégraphie garantit que les composants du système sont faiblement couplés, ce qui leur permet d'évoluer, d'être déployés et de scaler indépendamment les uns des autres, améliorant ainsi la résilience et l'agilité globale du système.[5]

La justification de ce modèle hybride réside dans sa capacité à résoudre une dichotomie fondamentale de l'architecture d'entreprise moderne. Les processus d'assurance, avec leurs exigences de conformité, de traçabilité et de gestion des exceptions, bénéficient énormément de la structure et de la visibilité offertes par un orchestrateur central.[1] Simultanément, la pression concurrentielle exige une architecture flexible, composée de services autonomes et spécialisés qui peuvent être développés, testés et mis à jour rapidement sans perturber l'ensemble de l'écosystème.[7] En combinant orchestration et chorégraphie, la PEA obtient le meilleur des deux mondes : un contrôle centralisé de la logique métier et une exécution décentralisée et résiliente.

Un diagramme architectural de haut niveau illustre cette interaction :

- **L'Orchestrateur (Camunda Platform 8)** se situe au centre, détenant la vision de bout en bout du processus (par exemple, une demande de souscription). Il ne contient pas de logique métier spécifique mais sait "qui" doit faire "quoi" et dans "quel ordre".
- **Le Bus d'Événements (Apache Kafka)** entoure l'orchestrateur, servant de canal de communication unique. Tous les échanges entre l'orchestrateur et les agents transitent par ce bus.
- **Les Agents (Workers Externes)** sont des services autonomes qui s'abonnent à des topics spécifiques sur le bus d'événements. Chaque agent est un expert dans un domaine précis (vérification de crédit, évaluation des risques, génération de documents).
- **Les Systèmes Externes (API tierces, bases de données, systèmes legacy)** sont accessibles uniquement par les agents, qui agissent comme des façades ou des adaptateurs, protégeant ainsi l'orchestrateur de la complexité des intégrations point à point.

1.2 Sélection de la Stack Technologique

Le choix des technologies fondamentales pour la PEA a été guidé par des principes de standardisation, de scalabilité "cloud-native" et de pérennité architecturale.

- **Camunda Platform 8 (Moteur d'Orchestration) :** La sélection de Camunda Platform 8 est motivée par son alignement avec les exigences d'une architecture moderne et distribuée. Son cœur, le moteur de workflow Zeebe, est conçu dès le départ pour être "cloud-native". Il s'agit d'un système distribué qui garantit la tolérance aux pannes et permet une scalabilité horizontale linéaire simplement en ajoutant des nœuds au cluster, une capacité essentielle pour gérer les pics de charge inhérents au secteur de l'assurance.[9] De plus, Camunda repose sur des standards ouverts et reconnus par l'industrie : BPMN 2.0 pour la modélisation des processus et DMN 1.3 pour l'automatisation des décisions. Cette adhésion aux standards favorise la création d'un langage commun entre les équipes métier et techniques, garantissant que la logique modélisée est précisément celle qui est exécutée, réduisant ainsi les ambiguïtés et accélérant les cycles de développement.[11] L'écosystème d'outils fourni, notamment Operate pour la surveillance et Tasklist pour la gestion des tâches humaines, offre des capacités opérationnelles prêtes à l'emploi qui sont cruciales pour la gestion de processus critiques.[9]
- **Apache Kafka via Confluent Cloud (Épine Dorsale Événementielle) :** Apache Kafka a été choisi comme bus d'événements pour ses performances exceptionnelles, sa faible latence, sa durabilité des messages et sa capacité à gérer des volumes de données massifs en temps réel.[13] Le choix de la version managée, Confluent Cloud, est une décision stratégique visant à déléguer la complexité opérationnelle de la gestion d'un cluster Kafka (provisionnement, mise à l'échelle, maintenance, sécurité) à un fournisseur expert. Cela permet aux équipes internes de se concentrer sur la création de valeur métier plutôt que sur la gestion de l'infrastructure.[15] Confluent Cloud enrichit Kafka avec des composants essentiels à une gouvernance d'entreprise robuste, notamment le Schema Registry pour la gestion des contrats de données et Kafka Connect pour une intégration simplifiée avec d'autres systèmes, qui sont des piliers de notre architecture.[16]

Le choix de cette stack technologique va au-delà d'une simple décision d'outillage ; il influence et façonne directement le modèle opérationnel de l'organisation. L'architecture "cloud-native" de Camunda 8, conçue pour être déployée via des conteneurs comme Docker et orchestrée par Kubernetes [10], s'aligne parfaitement avec les pratiques DevOps modernes. Le pattern "External Task", qui sera au cœur de notre modèle "agentique", découple la logique métier de l'orchestrateur. Cela permet aux équipes de développement de construire, tester et déployer leurs agents de manière totalement indépendante, favorisant une véritable appropriation du code ("you build it, you run it").[7] En parallèle, l'utilisation d'une plateforme managée comme Confluent Cloud s'intègre naturellement dans des pipelines d'intégration et de déploiement continus (CI/CD).[18] La combinaison de ces éléments crée un écosystème où des équipes produit autonomes peuvent posséder des microservices (leurs agents) de bout en bout, tout en opérant dans un cadre d'entreprise cohérent et gouverné par la plateforme d'orchestration. L'architecture n'est donc pas seulement une structure technique, mais un

catalyseur pour une organisation plus agile, décentralisée et alignée sur les principes des microservices.

2.0 Composants Centraux de la Plateforme

2.1 Moteur d'Orchestration : Camunda Platform 8

La plateforme Camunda 8 constitue le pilier de l'orchestration au sein de la PEA. Elle est composée de plusieurs éléments interdépendants qui, ensemble, fournissent une solution complète pour la modélisation, l'exécution et la surveillance des processus métier.

- **Moteur de Workflow (Zeebe) :** Zeebe est le composant central d'exécution. Contrairement aux moteurs de workflow monolithiques traditionnels, Zeebe est un système distribué basé sur une architecture de logs d'événements. Il assure la persistance de l'état de toutes les instances de processus de manière tolérante aux pannes. Sa conception lui permet de scaler horizontalement pour traiter un nombre virtuellement illimité de transactions avec une faible latence constante, ce qui est crucial pour les applications à haute charge.[9] Chaque action dans le processus est enregistrée comme un événement immuable, ce qui garantit une traçabilité et une résilience exceptionnelles.[10]
- **Modélisation de Processus (BPMN 2.0) :** Le standard *Business Process Model and Notation* (BPMN 2.0) sera le langage unique et formel pour décrire la logique de flux de tous les processus métier sur la plateforme. L'avantage fondamental de BPMN est sa nature visuelle et standardisée, qui le rend accessible à la fois aux analystes métier, aux chefs de produit et aux ingénieurs.[11] Cette notation graphique sert de contrat et de source unique de vérité, éliminant les décalages entre les spécifications fonctionnelles et l'implémentation technique. La philosophie de Camunda "what you model is what you run" garantit que le diagramme BPMN n'est pas une simple documentation, mais un artefact exécutable directement par le moteur Zeebe.[2]
- **Automatisation des Décisions (DMN 1.3) :** Pour la gestion des règles métier, la plateforme utilisera le standard *Decision Model and Notation* (DMN 1.3). DMN permet d'externaliser la logique de décision complexe (par exemple, les règles de tarification, les critères d'éligibilité, l'évaluation des risques) hors du code applicatif et des modèles de processus BPMN.[20] Ces règles sont modélisées dans des tables de décision claires et lisibles, que les experts du domaine peuvent comprendre et même modifier via des outils graphiques. Le moteur de décision de Camunda peut exécuter ces tables DMN nativement, ce qui permet de faire évoluer la logique métier avec une agilité bien plus

grande que si elle était codée en dur dans les applications.[21]

- **Suite Opérationnelle (Operate, Tasklist, Optimize) :** Camunda 8 est livré avec une suite d'applications web qui fournissent des capacités essentielles de surveillance et d'interaction.
  - **Operate :** Cet outil offre une visibilité en temps réel sur l'état de toutes les instances de processus en cours d'exécution. Il permet aux équipes opérationnelles de suivre les processus, d'identifier les goulots d'étranglement, d'analyser les erreurs et de résoudre les incidents directement depuis une interface graphique, sans nécessiter d'intervention technique approfondie.[9]
  - **Tasklist :** Cette application est l'interface dédiée aux tâches humaines. Lorsqu'un processus requiert une intervention manuelle (par exemple, une approbation, une validation de données ou la gestion d'un cas d'exception), une tâche est créée et apparaît dans la Tasklist de l'utilisateur ou du groupe approprié. C'est un composant indispensable pour les processus qui ne sont pas entièrement automatisés.[9]
  - **Optimize :** Optimize fournit des capacités d'analyse et de *process mining* sur les données historiques d'exécution. Il permet de générer des rapports, des tableaux de bord et des cartes de chaleur pour identifier les inefficacités, mesurer les indicateurs de performance clés (KPIs) et découvrir des opportunités d'amélioration continue des processus.[9]

2.2 Épine Dorsale Événementielle : Apache Kafka (Confluent Cloud)

L'épine dorsale événementielle, basée sur Apache Kafka et gérée via Confluent Cloud, est le système circulatoire de la PEA. Elle ne se contente pas de transporter des messages, mais assure également la gouvernance, la résilience et l'intégrabilité de l'écosystème.

- **Architecture des Topics :** Les topics Kafka sont les canaux de communication fondamentaux. Chaque topic sera dédié à un type d'événement métier spécifique et sémantiquement cohérent (par exemple, demande-de-souscription-soumise). Ils seront systématiquement partitionnés pour permettre une consommation parallèle par plusieurs instances d'un même agent, garantissant ainsi une haute disponibilité et une scalabilité du traitement.[13] Une convention de nommage stricte et hiérarchique sera appliquée pour assurer la clarté et la découvrabilité (détaillée dans la section 5.1).
- **Gouvernance des Données (Schema Registry) :** L'utilisation du Schema Registry de Confluent est un principe architectural **non négociable** de la PEA. Il agira comme le référentiel centralisé et la source unique de vérité pour tous les schémas de données des événements transitant par Kafka. Chaque message produit ou consommé devra être validé par rapport à un schéma enregistré (en format Avro ou Protobuf). Le Schema Registry gère le versioning des schémas et applique des règles de compatibilité (par exemple, BACKWARD_COMPATIBILITY) pour contrôler leur évolution.[16] Cela prévient les

erreurs d'exécution dues à des formats de données incompatibles, évite la corruption des données et garantit un contrat de données explicite entre les services.[25]

- **Intégration de Systèmes (Kafka Connect) :** Kafka Connect sera l'outil standard pour intégrer la plateforme avec des systèmes de données externes, qu'ils soient des sources (pour ingérer des données dans Kafka) ou des puits (pour exporter des données de Kafka). Il fournit un framework déclaratif, scalable et tolérant aux pannes, ainsi qu'un vaste écosystème de connecteurs pré-construits pour des systèmes courants comme les bases de données relationnelles (via JDBC), les data lakes (S3, HDFS), les moteurs de recherche (Elasticsearch) et de nombreuses API tierces.[17] L'utilisation de Kafka Connect minimise le besoin de développer du code d'intégration personnalisé, ce qui réduit les coûts et accélère la mise sur le marché de nouvelles intégrations.[17]

L'imposition du Schema Registry est une décision stratégique qui transforme Kafka d'un simple "tuyau de messages" en une véritable "colonne vertébrale de données" d'entreprise. Sans une gouvernance stricte des schémas, une architecture événementielle, à mesure qu'elle grandit, est vouée au chaos. Les consommateurs de messages se brisent de manière imprévisible chaque fois qu'un producteur modifie le format de ses événements, créant une dette technique et une fragilité systémique.[27] Le Schema Registry impose un contrat de données formel. Un producteur ne peut physiquement pas publier un message qui ne respecte pas le schéma convenu, et un consommateur sait avec certitude la structure des données qu'il va recevoir.[16] Les règles de compatibilité, comme

BACKWARD, sont le mécanisme qui rend l'évolution agile possible. Elles permettent à une équipe de faire évoluer le schéma d'un événement (par exemple, en ajoutant un nouveau champ optionnel) et de déployer son service producteur en toute confiance, sachant que les consommateurs existants ne seront pas affectés.[25] Cela découple fondamentalement les cycles de vie de déploiement des différentes équipes. L'équipe A peut déployer une nouvelle version de son agent sans avoir à orchestrer un déploiement "big bang" synchronisé avec les équipes B, C et D qui consomment ses événements.[27] Par conséquent, le Schema Registry n'est pas une simple fonctionnalité technique ; c'est le garant de l'agilité, du découplage et de la maintenabilité à long terme promis par l'architecture microservices.

Partie II : Le Modèle Agentique et le Projet Pilote

3.0 Le Modèle Agentique : Implémentation via le Pattern "External Task"

Le concept "agentique" de la plateforme est la manifestation concrète du principe de découplage entre l'orchestration et l'exécution. Il est implémenté techniquement par le pattern "External Task" de Camunda, qui transforme les tâches de service BPMN en éléments de travail asynchrones traités par des applications externes et autonomes.

3.1 Définition d'un "Agent"

Dans le contexte de la PEA, un "Agent" est un worker externe. Il s'agit d'une application indépendante, généralement un microservice, qui s'abonne à un "topic de travail" spécifique défini dans le modèle BPMN. Par exemple, une tâche de service BPMN nommée "Vérifier le score de crédit" sera configurée avec le type "External" et un nom de topic, tel que credit-score-check.[7] Les agents sont les effecteurs du système : ils contiennent et exécutent la logique métier concrète (appeler une API, effectuer un calcul, interagir avec une base de données), laissant à l'orchestrateur la seule responsabilité de la séquence du processus.[19]

Ce modèle offre des avantages architecturaux significatifs :

- **Polyglotte :** Les agents étant découplés via des API REST ou gRPC, ils peuvent être développés dans n'importe quel langage de programmation. Une équipe peut choisir Java/Spring Boot pour un agent nécessitant une intégration complexe avec des systèmes d'entreprise, tandis qu'une autre peut opter pour Python pour un agent effectuant des calculs de machine learning. Cela permet d'utiliser la technologie la plus appropriée pour chaque problème spécifique, sans imposer une stack technologique unique à toute l'organisation.[7]
- **Scalabilité Indépendante :** Chaque type d'agent peut être mis à l'échelle horizontalement en fonction de sa charge de travail spécifique. Si l'agent de génération de polices PDF devient un goulot d'étranglement en raison de sa consommation intensive de CPU, il est possible de déployer de multiples instances de cet agent uniquement, sans avoir à surdimensionner le moteur de processus ou d'autres agents moins sollicités.[7]
- **Résilience Accrue :** La défaillance d'un agent n'entraîne pas la défaillance de l'ensemble du processus. Si un agent tombe en panne ou ne parvient pas à traiter une tâche, celle-ci reste simplement dans la file d'attente de travail du moteur de processus. Une fois l'agent redémarré ou corrigé, il peut reprendre le traitement de la tâche là où il s'était arrêté, garantissant ainsi qu'aucun travail n'est perdu.[7]

3.2 Cycle de Vie d'une Tâche Externe

Le cycle de vie d'une interaction entre l'orchestrateur et un agent suit un protocole bien défini, garantissant une exécution asynchrone et fiable.

1. **Création :** Lorsque le moteur Zeebe atteint une "Service Task" configurée comme "External", il ne tente pas d'exécuter de code. À la place, il crée une entrée dans sa file d'attente de travail interne, associée au topic spécifié (par exemple, credit-score-check). L'instance de processus se met alors en état d'attente à cette étape précise.[23]
2. **Polling et Verrouillage (Fetch and Lock) :** Les agents correspondants interrogent activement (poll) le moteur de processus à intervalles réguliers, demandant s'il y a du travail disponible sur les topics auxquels ils sont abonnés. Lorsqu'un agent reçoit une ou plusieurs tâches, il les verrouille immédiatement pour une durée configurable. Ce verrouillage est un mécanisme de concurrence crucial : il garantit qu'une seule instance d'agent traitera une tâche donnée à un moment T, même si plusieurs instances du même agent sont en cours d'exécution.[8]
3. **Exécution :** L'agent exécute la logique métier pour laquelle il a été conçu. Il a accès à toutes les variables de processus nécessaires qui ont été spécifiées lors de la configuration de la souscription.
4. **Complétion :** Une fois la tâche terminée avec succès, l'agent envoie une commande de "complétion" au moteur de processus, en passant l'identifiant unique de la tâche. Il peut également joindre de nouvelles variables ou mettre à jour des variables existantes qui seront fusionnées dans l'état de l'instance de processus. À la réception de cette commande, le moteur déverrouille et supprime la tâche de la file d'attente, puis fait avancer l'instance de processus à l'étape suivante du modèle BPMN.[8]
5. **Gestion des Échecs et Incidents :** La gestion des erreurs est un aspect fondamental de ce modèle.
    - **Échec Technique (Failure) :** Si un agent rencontre une erreur technique transitoire (par exemple, une API tierce est temporairement indisponible, une connexion réseau échoue), il peut signaler un "échec" au moteur. Le moteur décrémentera alors un compteur de tentatives associé à la tâche. Le verrou expirera après la durée configurée, et la tâche redeviendra disponible pour être récupérée par un autre (ou le même) agent. Ce mécanisme de tentatives intégrées permet de gérer les pannes temporaires de manière robuste.[38]
    - **Erreur Métier (BPMN Error) :** Si l'agent rencontre une erreur métier attendue et gérable (par exemple, un client est jugé inéligible, des fonds sont insuffisants), il ne doit pas signaler un échec technique. Il doit plutôt lancer une "erreur BPMN" avec un code d'erreur spécifique. Cette action indique au moteur de ne pas retenter la tâche, mais de chercher dans le modèle BPMN un chemin d'exception (un "Error Boundary Event") correspondant à ce code d'erreur. Cela permet d'orchestrer la gestion des exceptions métier de manière explicite et visible dans le processus.[38]
    - **Incident :** Si une tâche épuise toutes ses tentatives sans succès, le moteur de processus crée un "incident". L'instance de processus est alors suspendue, et

l'incident est rendu visible dans Camunda Operate. Cela signale qu'une intervention humaine est nécessaire pour analyser la cause racine du problème (par exemple, un bug dans l'agent, des données corrompues) et décider de la marche à suivre (corriger les données et relancer la tâche, ou annuler le processus).[38]

3.3 Spécifications de Développement des Agents

Pour garantir la cohérence, la fiabilité et la maintenabilité des agents développés pour la PEA, les spécifications suivantes doivent être respectées.

- **Framework Recommandé :** Pour les agents développés sur la plateforme Java, l'utilisation du starter Spring Boot camunda-bpm-spring-boot-starter-external-task-client est la norme préconisée. Ce starter abstrait une grande partie de la complexité du cycle de vie (polling, verrouillage, complétion) et permet aux développeurs de se concentrer sur la logique métier. La souscription à un topic se fait simplement via une annotation @ExternalTaskSubscription("topic-name") sur une classe qui implémente l'interface ExternalTaskHandler.[42] Des exemples de projets sont disponibles pour guider les développeurs.[41]
- **Idempotence :** C'est une exigence critique pour tous les agents. Le protocole de communication entre le moteur et les agents garantit une livraison "au moins une fois" (*at-least-once*). Cela signifie qu'en cas de défaillance réseau ou de crash d'un agent après l'exécution de sa logique mais avant d'avoir pu envoyer la commande de complétion, le moteur considérera la tâche comme non terminée. Après l'expiration du verrou, la tâche sera réattribuée. L'agent doit donc être conçu pour pouvoir traiter la même tâche plusieurs fois sans provoquer d'effets de bord indésirables (par exemple, ne pas débiter un client deux fois). Des stratégies comme la vérification d'un identifiant de transaction unique avant d'exécuter une action sont nécessaires pour assurer l'idempotence.[8]

4.0 Projet Pilote : Processus de Souscription d'Assurance Automatisée

Le projet pilote a pour objectif de valider l'architecture de la PEA en implémentant un processus métier de bout en bout : la souscription à une police d'assurance automobile. Ce cas d'usage a été choisi pour sa complexité représentative, impliquant des règles métier, des intégrations externes, des décisions automatisées et des interventions humaines potentielles.

4.1 Modélisation du Processus Métier (BPMN)

Le processus sera formellement modélisé en BPMN 2.0, en utilisant des "pools" pour distinguer les responsabilités entre les différents systèmes et acteurs. Le flux principal, inspiré des processus d'assurance standards, suivra les étapes logiques ci-dessous [2] :

1. **Réception de la Demande :** Le processus est initié par un "Message Start Event" qui écoute un événement sur un topic Kafka, signalant qu'un client a soumis une nouvelle demande de souscription.
2. **Collecte de Données Initiales (Tâche de Service) :** Un premier agent est activé pour valider et structurer les données de la demande initiale.
3. **Vérification de l'Éligibilité (Tâche de Règle Métier) :** Le processus invoque une table de décision DMN pour effectuer un premier filtrage rapide basé sur des critères non négociables (âge du conducteur, type de véhicule interdit, pays de résidence). Si le demandeur n'est pas éligible, le processus est terminé avec une notification de rejet.
4. **Enrichissement des Données (Tâches de Service en Parallèle) :** Une passerelle parallèle est utilisée pour lancer simultanément plusieurs tâches d'enrichissement de données, chacune gérée par un agent spécialisé :
    - **Agent de Vérification de Crédit :** Interroge une agence de crédit externe pour obtenir le score de crédit du demandeur.
    - **Agent de Vérification de l'Historique du Véhicule :** Utilise le numéro d'identification du véhicule (VIN) pour interroger une base de données sur l'historique des accidents et des sinistres.
    - **Agent de Détection de Fraude :** Soumet les données du demandeur à un service interne ou externe de détection de fraude.
5. **Évaluation Complète des Risques (Tâche de Règle Métier) :** Après la synchronisation des tâches parallèles, une table de décision DMN plus complexe est invoquée. Elle prend en entrée l'ensemble des données enrichies pour calculer un score de risque composite.
6. **Passerelle Exclusive (Décision de Souscription) :** Basé sur le score de risque calculé, une passerelle exclusive route le processus vers l'un des trois chemins suivants :
    - **Approbation Automatique :** Si le score de risque est en dessous d'un seuil défini, la demande est approuvée sans intervention humaine (*straight-through processing*).
    - **Rejet Automatique :** Si le score de risque est au-dessus d'un seuil critique, la demande est automatiquement rejetée.
    - **Examen Manuel :** Si le score se situe dans une "zone grise", le dossier est transmis à un expert humain pour une analyse plus approfondie.
7. **Examen par le Souscripteur (Tâche Utilisateur) :** Dans le cas d'un examen manuel, une tâche utilisateur est créée dans la Tasklist de Camunda. Un souscripteur qualifié examine toutes les données collectées et prend une décision finale (Approuver ou Rejeter).
8. **Calcul de la Prime (Tâche de Service) :** Si la demande est approuvée

(automatiquement ou manuellement), un agent de tarification est invoqué pour calculer le montant exact de la prime d'assurance.

9. **Génération de la Police (Tâche de Service) :** Un agent de génération de documents crée le contrat de police officiel au format PDF, en y insérant toutes les informations pertinentes.
10. **Notification du Client (Tâche de Service) :** Un agent de notification envoie un email au client. En cas d'approbation, l'email contient la police en pièce jointe. En cas de rejet, il contient la lettre de notification correspondante.
11. **Fin du Processus.**

4.2 Automatisation des Décisions (DMN)

La logique de décision du processus sera encapsulée dans un catalogue de tables de décision DMN, permettant une gestion agile et transparente des règles métier.[21]

- **Catalogue des Décisions :**
  - **eligibilityCheck :** Une table simple pour les vérifications de base. Elle utilisera une politique de "Hit" de type UNIQUE, car une seule règle d'éligibilité doit s'appliquer.
  - **riskAssessment :** Une table de décision complexe qui évalue de multiples facteurs de risque. Elle utilisera une politique de "Hit" de type COLLECT (SUM). Cela signifie que plusieurs règles peuvent être satisfaites simultanément (par exemple, un jeune conducteur ET un véhicule puissant), et le résultat final sera la somme des points de risque de toutes les règles déclenchées. Cette approche permet de construire un score de risque granulaire et composite.[48]
  - **premiumCalculation :** Une table qui mappe le score de risque final et d'autres facteurs (options de couverture, franchise) à une prime de base. Elle utilisera une politique de "Hit" de type FIRST, car une seule prime de base doit être sélectionnée.
- **Exemple de Règle (dans la table riskAssessment) :**
  - **Inputs :** applicant.age < 21, vehicle.enginePower > 250, credit.score < 620
  - **Input Expressions :** `
- **Communication Orchestrateur -> Agent :** Ce flux est géré de manière standard par le pattern "External Task" décrit précédemment. Camunda crée une tâche, et l'agent la récupère.
- **Communication Agent -> Orchestrateur (Request/Response Asynchrone) :** Ce pattern est utilisé lorsque le processus doit attendre une réponse d'un agent avant de continuer. L'agent ne répond pas directement au moteur via une API synchrone. Au lieu de cela, il publie un événement de résultat sur un topic Kafka de réponse dédié (par exemple, insurance.subscription.datarichment-completed.v1). Dans le modèle BPMN, un "Intermediate Message Catch Event" est placé après la tâche de service. Cet événement est configuré avec un connecteur Kafka entrant qui écoute sur le topic de réponse. La

corrélation entre cet événement et la bonne instance de processus en attente est le défi central de ce pattern. Elle est résolue en utilisant un identifiant unique (par exemple, applicationId) qui est propagé tout au long du flux. L'événement Kafka doit contenir cet identifiant, et l'événement de réception dans BPMN est configuré pour utiliser cette valeur afin de "réveiller" l'instance de processus correspondante et aucune autre.[51]

La fiabilité de ce mécanisme de corrélation est primordiale pour le bon fonctionnement de l'architecture. Une stratégie de gestion des identifiants de corrélation mal conçue peut conduire à des processus bloqués (un événement de réponse n'est jamais corrélé) ou à des erreurs de traitement (un événement est corrélé à la mauvaise instance). Par conséquent, la plateforme doit imposer une règle stricte : tout échange asynchrone qui nécessite une réponse doit contenir un correlationId unique et traçable, généré au début du processus. Cet identifiant doit être présent dans les en-têtes ou le payload de chaque message Kafka. Le moteur Camunda est ensuite configuré pour utiliser une expression de corrélation (par exemple, = kafkaMessage.key ou = kafkaMessage.value.applicationId) pour faire correspondre les messages entrants aux instances de processus en attente.[50] Cette discipline architecturale est la clé pour garantir la robustesse et la fiabilité d'un système distribué et asynchrone.

Partie III : Gouvernance, Sécurité et Exigences Transverses

5.0 Gouvernance des Données et des Événements

Une gouvernance rigoureuse des données et des événements est essentielle pour assurer la cohérence, la maintenabilité et l'évolutivité de la Plateforme d'Entreprise Agentique. Elle repose sur trois piliers : des conventions de nommage strictes, des schémas de données formels et une stratégie d'évolution contrôlée.

5.1 Conventions de Nommage pour les Topics Kafka

Pour éviter la prolifération anarchique de topics et pour garantir la clarté, la découvrabilité et la gestion fine des permissions, une convention de nommage hiérarchique et standardisée

sera appliquée à tous les topics Kafka.[18]

- **Format :** <domaine>.<sous-domaine>.<type-evenement>.<version>
  - **<domaine> :** Le domaine métier de haut niveau (par exemple, insurance, finance, customer).
  - **<sous-domaine> :** Un sous-domaine fonctionnel spécifique (par exemple, subscription, claim, billing).
  - **<type-evenement> :** Une description concise de l'événement, utilisant un verbe au participe passé (par exemple, application-submitted, risk-assessment-completed).
  - **<version> :** La version du schéma de l'événement (par exemple, v1, v2).
- **Exemples pour le Projet Pilote :**
  - insurance.subscription.application-submitted.v1
  - insurance.subscription.credit-check-completed.v1
  - insurance.subscription.risk-assessment-completed.v1
  - insurance.subscription.policy-issued.v1
  - insurance.subscription.application-rejected.v1

Cette structure permet de configurer des permissions granulaires via des wildcards (par exemple, autoriser un agent à lire tous les événements du sous-domaine insurance.subscription.*.v1).

5.2 Spécification des Schémas d'Événements

- **Format de Sérialisation :** Apache Avro est désigné comme le format de sérialisation standard pour tous les événements. Ce choix est motivé par sa nature binaire compacte (qui réduit l'utilisation de la bande passante et du stockage), ses capacités natives et robustes d'évolution de schéma, et son intégration transparente avec le Confluent Schema Registry.[25]
- **Définition des Schémas :** Chaque événement aura un schéma formellement défini dans un fichier .avsc. Ce schéma décrira la structure complète du payload de l'événement, en spécifiant les noms des champs, leurs types de données (primitifs et complexes), les valeurs par défaut pour les champs optionnels, et une documentation claire pour chaque champ.[25]
- **Dictionnaire des Événements :** Un dictionnaire central des événements sera maintenu pour servir de contrat de données formel pour l'ensemble de la plateforme. Ce document est indispensable pour la coordination entre les équipes de développement et pour la gouvernance globale des données.

Nom du Topic	Nom du Schéma	Version	Description de	Champs Clés du	Producteurs	Consommateurs

	(Subject)		l'Événement	Payload	Autorisés	Autorisés
insurance.subscription.application-submitted.v1	application-submitted-v1	1	Un nouveau client a soumis une demande de souscription.	applicationId, customerDetails, vehicleDetails	WebApp Gateway	Camunda Orchestrator
insurance.subscription.risk-assessment-completed.v1	risk-assessment-completed-v1	1	L'évaluation des risques pour une demande est terminée.	applicationId, riskScore, riskFactors	RiskAssessmentAgent	Camunda Orchestrator
insurance.subscription.policy-issued.v1	policy-issued-v1	1	Une police d'assurance a été générée et émise.	applicationId, policyNumber, premiumAmount, documentUrl	DocumentGenerationAgent	NotificationAgent, BillingSystemConnector
Table 2 : Dictionnaire des Événements Kafka du Projet Pilote						

5.3 Stratégie d'Évolution des Schémas

Pour permettre une évolution agile et découplée des services, la règle de compatibilité des schémas par défaut, appliquée par le Schema Registry, sera BACKWARD_COMPATIBILITY. Cette règle stipule qu'un schéma peut évoluer (par exemple, en ajoutant un nouveau champ optionnel avec une valeur par défaut, ou en supprimant un champ optionnel) tant que les consommateurs utilisant une version plus ancienne du schéma peuvent toujours lire et désérialiser les messages produits avec la nouvelle version. Cette stratégie est fondamentale car elle permet de mettre à jour et de déployer des services producteurs sans nécessiter une mise à jour simultanée de tous leurs consommateurs, évitant ainsi des dépendances de déploiement rigides et des interruptions de service.[25]

6.0 Sécurité, Conformité et Auditabilité

La sécurité est une exigence fondamentale de la PEA, intégrée à chaque couche de l'architecture. Elle couvre la gestion des identités, la sécurisation des communications et la capacité d'audit complète des activités.

6.1 Gestion des Identités et des Accès (IAM)

- **Single Sign-On (SSO) pour les Outils Camunda :** L'accès aux interfaces web de Camunda (Operate, Tasklist, Modeler) sera sécurisé par une intégration avec le fournisseur d'identité (IdP) de l'entreprise via le protocole OpenID Connect (OIDC). Cela garantit que seuls les utilisateurs authentifiés peuvent accéder à ces outils, en s'appuyant sur les mécanismes d'authentification centralisés et sécurisés de l'entreprise (par exemple, l'authentification multifacteur).[57] L'identité de l'utilisateur sera résolue et exposée à travers les composants pour une meilleure traçabilité.[60]
- **Autorisation dans Camunda :** Les permissions au sein de la plateforme Camunda (par exemple, qui peut voir quels processus, qui peut agir sur quelles tâches) seront gérées par le système d'autorisation de Camunda. Les rôles et les groupes seront mappés à partir des "claims" (attributs) contenus dans le jeton OIDC fourni par l'IdP, permettant une gestion centralisée des habilitations au niveau de l'annuaire d'entreprise.[59]
- **Sécurisation de Kafka :** L'accès aux ressources Kafka (topics, consumer groups) sera rigoureusement contrôlé à l'aide de Listes de Contrôle d'Accès (ACLs). Chaque agent et chaque application sera représenté par un "principal" unique (par exemple, un compte de

service). Des ACLs spécifiques seront définies pour accorder à chaque principal uniquement les permissions minimales requises pour son fonctionnement (principe du moindre privilège). Par exemple, le PricingAgent aura la permission READ sur le topic des demandes de tarification et WRITE sur le topic des résultats, mais aucune permission sur les autres topics.[62]

6.2 Sécurisation des Communications

- **Chiffrement en Transit :** Toutes les communications réseau entre les composants de la plateforme doivent être chiffrées à l'aide du protocole Transport Layer Security (TLS 1.2 ou supérieur). Cela inclut les connexions des clients vers le moteur Camunda, les communications entre les nœuds du cluster Zeebe, les connexions des producteurs et consommateurs vers les brokers Kafka, et les communications inter-brokers au sein du cluster Kafka. Le chiffrement en transit garantit la confidentialité et l'intégrité des données, les protégeant contre l'interception et la manipulation.[62]
- **Gestion des Secrets :** Aucune information sensible (mots de passe, clés d'API, secrets de client OIDC, certificats TLS) ne doit être stockée en clair dans les fichiers de configuration, le code source ou les images de conteneurs. Toutes ces informations doivent être gérées comme des "secrets" et stockées dans un système de gestion de secrets centralisé et sécurisé (par exemple, HashiCorp Vault, AWS Secrets Manager, Azure Key Vault ou Google Secret Manager). Les applications et agents récupéreront ces secrets dynamiquement au moment de leur démarrage, en s'authentifiant auprès du gestionnaire de secrets via des identités de service managées.[15]

6.3 Journalisation et Audit

La plateforme est conçue pour offrir une auditabilité complète, permettant de reconstruire l'historique de chaque transaction de bout en bout.

- **Audit Trail de Camunda :** Le moteur de processus Camunda enregistre un historique détaillé de chaque instance de processus. Cet historique, conservé dans un stockage optimisé, contient chaque événement de workflow : le démarrage et la fin du processus, l'activation et la complétion de chaque tâche, les variables créées ou modifiées, et les décisions prises par les tables DMN. Cette piste d'audit native fournit une preuve irréfutable du déroulement du processus, essentielle pour la conformité réglementaire et la résolution de litiges.
- **Audit Logs de Kafka :** La fonctionnalité d'audit de Kafka sera activée pour enregistrer

chaque tentative d'accès aux ressources Kafka. Ces logs capturent des informations critiques pour la sécurité : quel principal a tenté d'effectuer quelle opération (par exemple, READ, WRITE), sur quelle ressource (par exemple, topic insurance.subscription.application-submitted.v1), à quel moment, et si l'opération a été autorisée ou refusée par les ACLs. Ces logs d'audit seront exportés et centralisés dans un système de gestion des informations et des événements de sécurité (SIEM) comme Splunk ou une stack ELK (Elasticsearch, Logstash, Kibana) pour la surveillance en temps réel des activités suspectes et l'analyse forensique en cas d'incident de sécurité.[65]

La combinaison des pistes d'audit de Camunda et de Kafka crée une capacité de traçabilité qui transcende les simples exigences techniques de conformité. Elle devient un puissant outil métier. Le moteur Camunda enregistre le "quoi" et le "pourquoi" de la logique métier : quelle décision a été prise, et sur la base de quelles données.[9] Simultanément, les logs d'audit de Kafka enregistrent le "qui", le "où" et le "quand" des accès aux données : quel service a lu ou écrit sur quel topic, et à quel moment précis.[65] En corrélant ces deux sources d'information via un identifiant de transaction unique (le

correlationId propagé à travers le système), il devient possible de reconstruire l'historique complet et inviolable d'une demande client à travers un écosystème de services distribués et asynchrones. Cette capacité permet de répondre avec une certitude absolue à des questions métier critiques : "Quel était l'état exact de toutes les variables lorsque la décision de rejet a été prise pour le client X?", "Quels sont tous les systèmes qui ont accédé aux données personnelles de ce client durant le processus de souscription?". Cette traçabilité granulaire est un atout majeur pour la gestion des risques, l'amélioration du service client (en fournissant des explications claires sur les décisions) et la démonstration de la conformité à des réglementations strictes comme le RGPD.

7.0 Exigences Non Fonctionnelles (NFRs)

Les exigences non fonctionnelles définissent les attributs de qualité de la plateforme. Elles sont aussi critiques que les exigences fonctionnelles et doivent être mesurables et testables pour garantir que le système répond aux attentes en matière de performance, de fiabilité et de maintenabilité.[68]

7.1 Performance et Latence

- **NFR-PERF-01 :** Le temps de traitement de bout en bout pour un processus de

souscription entièrement automatisé ("straight-through processing") doit être inférieur à 2 secondes, du moment de la réception de l'événement de soumission à l'envoi de la notification finale au client.
- **NFR-PERF-02 :** La latence d'évaluation d'une table de décision DMN par le moteur de décision doit être inférieure à 50 millisecondes au 99ème percentile.
- **NFR-PERF-03 :** La latence de publication d'un message sur un topic Kafka (temps de reconnaissance par le broker) doit être inférieure à 10 millisecondes au 99ème percentile.

7.2 Scalabilité

- **NFR-SCAL-01 :** La plateforme doit être capable de traiter un pic de 10 000 nouvelles demandes de souscription par heure sans dégradation des performances définies dans la section 7.1.
- **NFR-SCAL-02 :** Chaque composant de la plateforme (cluster Zeebe, cluster Kafka, pools d'agents) doit pouvoir être mis à l'échelle horizontalement de manière indépendante pour répondre à une augmentation de la charge. Le système doit supporter une augmentation de 3x du nombre de nœuds ou d'instances sans nécessiter de refonte architecturale.

7.3 Haute Disponibilité et Résilience

- **NFR-HA-01 :** La plateforme dans son ensemble doit atteindre un objectif de disponibilité de 99,95%, correspondant à un temps d'arrêt maximal de 4,38 heures par an.
- **NFR-HA-02 :** Le système doit être déployé dans une configuration redondante sur plusieurs zones de disponibilité (Multi-AZ) pour survivre à la défaillance complète d'une zone.
- **NFR-HA-03 :** Aucune perte de données (état des processus ou messages Kafka) ne sera tolérée en cas de défaillance d'un nœud unique au sein des clusters Camunda ou Kafka. Les mécanismes de réplication et de basculement doivent être automatiques.

7.4 Maintenabilité et Observabilité

- **NFR-MNT-01 :** Tous les agents et composants de la plateforme doivent exposer des métriques de santé et de performance (par exemple, latence, débit, taux d'erreur) dans un format compatible avec Prometheus.
- **NFR-MNT-02 :** Toutes les sorties de logs doivent être au format JSON structuré pour

permettre une collecte, une indexation et une analyse centralisées efficaces.

- **NFR-MNT-03 :** Le traçage distribué utilisant le standard OpenTelemetry doit être implémenté. Chaque requête ou événement doit porter un identifiant de trace qui permet de suivre son parcours à travers les différents agents et composants de la plateforme.

La formalisation de ces NFRs dans une matrice permet de transformer des objectifs qualitatifs ("le système doit être rapide et fiable") en critères quantifiables et vérifiables. Cette matrice sert de contrat entre les parties prenantes et constitue la base pour la définition des stratégies de test de performance, de charge et de résilience.

Catégorie	ID de l'Exigence	Description de l'Exigence	Métrique	Objectif	Méthode de Validation	Priorité
Performance	NFR-PERF-01	Temps de traitement de bout en bout (STP)	Temps de traitement (P95)	< 2 secondes	Test de charge de bout en bout	Critique
Performance	NFR-PERF-02	Latence d'évaluation DMN	Temps de réponse (P99)	< 50 ms	Test de performance unitaire	Élevée
Scalabilité	NFR-SCAL-01	Capacité de traitement des demandes	Débit maximal soutenu	10 000 demandes/heure	Test de charge en pic	Critique
Disponibilité	NFR-HA-01	Disponibilité du service	Uptime mensuel	99,95%	Surveillance continue	Critique
Disponibilité	NFR-HA-02	Tolérance aux pannes de zone	Temps de récupération (RTO)	< 5 minutes	Test de chaos (Chaos Engineeri	Critique

					ng)	
Observabilité	NFR-MNT-03	Traçabilité des requêtes	Pourcentage de requêtes tracées	100%	Validation en environnement de test	Élevée
Table 3 : Matrice des Exigences Non Fonctionnelles (NFRs)						

Conclusion

Le présent document établit les spécifications fonctionnelles et architecturales de la Plateforme d'Entreprise Agentique (PEA), Version 1.0. L'architecture hybride proposée, qui allie l'orchestration de processus centralisée avec Camunda 8 et la chorégraphie événementielle asynchrone avec Apache Kafka, est conçue pour répondre aux défis du secteur de l'assurance : la nécessité d'une gouvernance rigoureuse et d'une agilité opérationnelle.

Le modèle "agentique", implémenté via le pattern "External Task", constitue le cœur de cette agilité. Il favorise la création d'un écosystème de microservices autonomes, polyglottes et scalables, tout en maintenant une vision de bout en bout et un contrôle centralisé des processus métier. Le projet pilote de souscription d'assurance automatisée servira de validation concrète de ces principes, en démontrant la capacité de la plateforme à gérer des flux de travail complexes, à automatiser des décisions basées sur des règles métier (DMN), et à s'intégrer de manière fiable avec des systèmes externes.

La gouvernance des données, la sécurité et les exigences non fonctionnelles ne sont pas des considérations secondaires mais des piliers fondamentaux de la plateforme. L'application stricte de conventions de nommage, l'utilisation obligatoire du Schema Registry, et une stratégie de sécurité multicouche (IAM, TLS, gestion des secrets) sont des prérequis non

négociables pour garantir la robustesse, la sécurité et la maintenabilité à long terme du système.

En conclusion, la PEA représente plus qu'une simple mise à niveau technologique. C'est une fondation stratégique conçue pour catalyser la transformation numérique de l'entreprise, en permettant le développement rapide de processus automatisés, résilients et intelligents, tout en fournissant les garanties de contrôle, de conformité et d'auditabilité indispensables à notre industrie. La mise en œuvre réussie de ce projet pilote ouvrira la voie à la migration progressive d'autres processus critiques sur cette nouvelle plateforme d'entreprise.

Ouvrages cités

1. Choosing the Right Integration Pattern for Your Business Processes - Camunda, dernier accès : octobre 1, 2025, https://camunda.com/blog/2023/07/choosing-the-right-integration-pattern-for-your-business-processes/
2. Personal claims process at an insurance company: BPMN process ..., dernier accès : octobre 1, 2025, https://www.researchgate.net/figure/Personal-claims-process-at-an-insurance-company-BPMN-process-diagram_fig11_266894308
3. Event-Driven Orchestration : Sample implementation | by Brijesh Deb | Medium, dernier accès : octobre 1, 2025, https://medium.com/@brijesh_deb/event-driven-orchestration-sample-implementation-187858e4031e
4. What is event-driven process orchestration? - Cognixia, dernier accès : octobre 1, 2025, https://www.cognixia.com/blog/what-is-event-driven-process-orchestration/
5. Event-Driven Architecture - AWS, dernier accès : octobre 1, 2025, https://aws.amazon.com/event-driven-architecture/
6. Event-Driven Process Orchestration: A Practitioner's Viewpoint - DZone, dernier accès : octobre 1, 2025, https://dzone.com/articles/event-driven-process-orchestration-a-practitioners
7. Maximizing Camunda's Potential with External Task Pattern - Content Services, dernier accès : octobre 1, 2025, https://contentservices.asee.io/maximizing-camundas-potential-with-external-task-pattern/
8. Camunda external tasks — a powerful tool for creating applications with a fault-tolerant and scalable architecture | by Alexandr Kazachenko | IT-компании Тинькофф | Medium, dernier accès : octobre 1, 2025, https://medium.com/its-tinkoff/camunda-external-tasks-a-powerful-tool-for-creating-applications-with-a-fault-tolerant-and-329b5ac3e1a6
9. Using Camunda 8, dernier accès : octobre 1, 2025, https://docs.camunda.io/docs/components/
10. How Camunda 8 Simplifies Workflow Automation for Java Teams, dernier accès : octobre 1, 2025,

https://camunda.com/blog/2024/12/how-camunda-8-simplifies-workflow-automation-for-java-teams/
11. BPMN Tutorial: Learn Business Process Model and Notation - Camunda, dernier accès : octobre 1, 2025, https://camunda.com/bpmn/
12. Agentic Orchestration & Automation Platform - Camunda, dernier accès : octobre 1, 2025, https://camunda.com/platform/
13. Documentation - Apache Kafka, dernier accès : octobre 1, 2025, https://kafka.apache.org/documentation/
14. Apache Kafka Use Cases [with Kafka Architecture Diagrams] - SoftKraft, dernier accès : octobre 1, 2025, https://www.softkraft.co/apache-kafka-use-cases/
15. Orchestration with Camunda and Kafka Confluent Cloud | Camunda, dernier accès : octobre 1, 2025, https://camunda.com/blog/2023/11/orchestration-camunda-kafka/
16. Schema Registry for Confluent Platform, dernier accès : octobre 1, 2025, https://docs.confluent.io/platform/current/schema-registry/index.html
17. Kafka Connect | Confluent Documentation, dernier accès : octobre 1, 2025, https://docs.confluent.io/platform/current/connect/index.html
18. Apache Kafka: Topic Naming Conventions - DEV Community, dernier accès : octobre 1, 2025, https://dev.to/devshawn/apache-kafka-topic-naming-conventions-3do6
19. Writing Good Workers For Camunda Cloud | by Bernd Rücker - berndruecker, dernier accès : octobre 1, 2025, https://blog.bernd-ruecker.com/writing-good-workers-for-camunda-cloud-61d322cad862
20. DMN Decision Engine - Camunda, dernier accès : octobre 1, 2025, https://camunda.com/platform/decision-engine/
21. What is DMN? DMN Tutorial and Examples | Camunda, dernier accès : octobre 1, 2025, https://camunda.com/dmn/
22. Create decision tables using DMN - Camunda 8 Docs, dernier accès : octobre 1, 2025, https://docs.camunda.io/docs/guides/create-decision-tables-using-dmn/
23. Service Task | Camunda Exercises, dernier accès : octobre 1, 2025, https://camunda-university-meetup.github.io/exercises/camunda-8/tasks/service-task/
24. The Future of User Tasks in Camunda 8: Moving Beyond Job Workers | by Tarek Mebrouk, dernier accès : octobre 1, 2025, https://medium.com/@tarekmebrouk/the-future-of-user-tasks-in-camunda-8-moving-beyond-job-workers-0eab547a2f38
25. Understanding Schema Registry in Kafka: Why It's Crucial for Data, dernier accès : octobre 1, 2025, https://medium.com/@DevBox_rohitrawat/understanding-schema-registry-in-kafka-why-its-crucial-for-data-integrity-and-scalability-3e8cf23179d5
26. Comprehensive Guide to Kafka Schema Registry - RisingWave, dernier accès : octobre 1, 2025, https://risingwave.com/blog/comprehensive-guide-to-kafka-schema-registry/
27. How Kafka Schema Registry Works and Why You Should Use It - Siva Prasad Rao

Janapati, dernier accès : octobre 1, 2025, https://smarttechie.co/how-kafka-schema-registry-works-and-why-you-should-use-it-9297d7538e4e
28. Kafka Connect Architecture - Confluent Documentation, dernier accès : octobre 1, 2025, https://docs.confluent.io/platform/current/connect/design.html
29. Introduction to Kafka Connect | Red Hat Developer, dernier accès : octobre 1, 2025, https://developers.redhat.com/articles/2023/12/27/introduction-kafka-connect
30. What is Kafka Connect? Concepts & Best Practices - AutoMQ, dernier accès : octobre 1, 2025, https://www.automq.com/blog/kafka-connect-architecture-concepts-best-practices
31. Confluent Cloud Fully-Managed Managed Connectors | Confluent Documentation, dernier accès : octobre 1, 2025, https://docs.confluent.io/cloud/current/connectors/overview.html
32. Confluent Connector Portfolio, dernier accès : octobre 1, 2025, https://www.confluent.io/product/connectors/
33. Easily Connect Data Systems in Real Time With Confluent, dernier accès : octobre 1, 2025, https://www.confluent.io/product/confluent-connectors/
34. All you Need to Know to Use Kafka Connectors - Hevo Data, dernier accès : octobre 1, 2025, https://hevodata.com/learn/kafka-connectors/
35. Kafka Schema Registry: When is it Really Necessary? : r/apachekafka - Reddit, dernier accès : octobre 1, 2025, https://www.reddit.com/r/apachekafka/comments/1jmsshi/kafka_schema_registry_when_is_it_really_necessary/
36. Camunda External Task Worker - The Basics - viadee Blog, dernier accès : octobre 1, 2025, https://blog.viadee.de/en/camunda-external-task-worker
37. Service Task | docs.camunda.org, dernier accès : octobre 1, 2025, https://docs.camunda.org/manual/latest/reference/bpmn20/tasks/service-task/
38. External Task Client | docs.camunda.org, dernier accès : octobre 1, 2025, https://docs.camunda.org/manual/latest/user-guide/ext-client/
39. Service tasks | Camunda 8 Docs, dernier accès : octobre 1, 2025, https://docs.camunda.io/docs/components/modeler/bpmn/service-tasks/
40. Incidents | Camunda 8 Docs, dernier accès : octobre 1, 2025, https://docs.camunda.io/docs/components/concepts/incidents/
41. Camunda process orchestration with external Java tasks. | by Hanno Van Der Walt | Medium, dernier accès : octobre 1, 2025, https://medium.com/@hannovdw1998/camunda-process-orchestration-with-external-java-tasks-ab27b497663e
42. External Task Client Spring Boot Starter | docs.camunda.org, dernier accès : octobre 1, 2025, https://docs.camunda.org/manual/latest/user-guide/ext-client/spring-boot-starter/
43. Camunda External Task Implementation : Detailed Explanation | by Harish Gupta | Medium, dernier accès : octobre 1, 2025,

https://medium.com/@gupta.harish2624/camunda-external-task-implementation-detailed-explanation-38f87527edf6
44. Spring Boot Starter for the External Task Client - Camunda, dernier accès : octobre 1, 2025, https://camunda.com/blog/2021/03/external-task-client-spring-bootified/
45. camunda-consulting/external-task-worker-examples - GitHub, dernier accès : octobre 1, 2025, https://github.com/camunda-consulting/external-task-worker-examples
46. A BPMN model illustrating the car insurance claim process. - ResearchGate, dernier accès : octobre 1, 2025, https://www.researchgate.net/figure/A-BPMN-model-illustrating-the-car-insurance-claim-process_fig2_286650392
47. BPM Insurance Strategies: 4 Steps to Better Processes, dernier accès : octobre 1, 2025, https://appian.com/blog/acp/insurance/bpm-for-insurance
48. DMN Hit Policy | docs.camunda.org, dernier accès : octobre 1, 2025, https://docs.camunda.org/manual/latest/reference/dmn/decision-table/hit-policy/
49. How Decision Modeling and AI Help with Risk Assessment - Camunda, dernier accès : octobre 1, 2025, https://camunda.com/blog/2024/07/how-decision-modeling-and-ai-help-with-risk-assessment/
50. Kafka connector | Camunda 8 Docs, dernier accès : octobre 1, 2025, https://docs.camunda.io/docs/components/connectors/out-of-the-box-connectors/kafka/
51. Using Kafka Connectors in Camunda for Request-Response Patterns with Concurrent Consumers - DEV Community, dernier accès : octobre 1, 2025, https://dev.to/devaaai/using-kafka-connectors-in-camunda-for-request-response-patterns-with-concurrent-consumers-4k9e
52. Service integration patterns with BPMN - Camunda 8 Docs, dernier accès : octobre 1, 2025, https://docs.camunda.io/docs/components/best-practices/development/service-integration-patterns/
53. Service Integration Patterns With BPMN And Camunda Cloud | by Bernd Rücker, dernier accès : octobre 1, 2025, https://blog.bernd-ruecker.com/service-integration-patterns-with-bpmn-and-camunda-cloud-53b0f458e49
54. Kafka Topic Naming Conventions: Best Practices, Patterns, and Guidelines - Confluent, dernier accès : octobre 1, 2025, https://www.confluent.io/learn/kafka-topic-naming-convention/
55. Event Design and Event Streams Best Practices - Confluent Developer, dernier accès : octobre 1, 2025, https://developer.confluent.io/courses/event-design/best-practices/
56. What is the best practice for naming kafka topics? - Stack Overflow, dernier accès : octobre 1, 2025, https://stackoverflow.com/questions/43726571/what-is-the-best-practice-for-naming-kafka-topics

57. OpenID Connect best security practices - Getting Started - Criipto Verify Documentation, dernier accès : octobre 1, 2025, https://docs.criipto.com/verify/getting-started/best-security-practices/
58. Connect to an OpenID Connect provider | Camunda 8 Docs, dernier accès : octobre 1, 2025, https://docs.camunda.io/docs/self-managed/identity/configuration/connect-to-an-oidc-provider/
59. Spring Security OAuth2 Integration | docs.camunda.org, dernier accès : octobre 1, 2025, https://docs.camunda.org/manual/latest/user-guide/spring-boot-integration/spring-security/
60. Enhanced Identity Resolution for OIDC Users - Camunda Product Roadmap, dernier accès : octobre 1, 2025, https://roadmap.camunda.com/c/266-enhanced-identity-resolution-for-oidc-users
61. Q&A: The One with the SSO Implementation in Camunda, dernier accès : octobre 1, 2025, https://camunda.com/blog/2021/11/qa-the-one-with-the-sso-implementation-in-camunda/
62. Kafka Logging Guide: Securing Kafka Logs - CrowdStrike, dernier accès : octobre 1, 2025, https://www.crowdstrike.com/en-us/guides/kafka-logging/securing-kafka-logs/
63. Kafka Security Auditing: Tools and Techniques | meshIQ Blog, dernier accès : octobre 1, 2025, https://www.meshiq.com/kafka-security-auditing-tools-and-techniques/
64. Ensuring Data Integrity in Kafka Audit Logging Practices | MoldStud, dernier accès : octobre 1, 2025, https://moldstud.com/articles/p-the-importance-of-data-integrity-in-kafka-audit-logging-ensuring-reliable-and-secure-data-management
65. Audit log concepts for Confluent Platform, dernier accès : octobre 1, 2025, https://docs.confluent.io/platform/current/security/compliance/audit-logs/audit-logs-concepts.html
66. Kafka Audit Logs with Log4j Plus ELK for Log Analysis - Confluent Developer, dernier accès : octobre 1, 2025, https://developer.confluent.io/courses/security/audit-logs/
67. Auditing Kafka - Event Streams - IBM, dernier accès : octobre 1, 2025, https://ibm.github.io/event-automation/es/administering/auditing-kafka/
68. Architecture 101: Top 10 Non-Functional Requirements (NFRs) you Should be Aware of, dernier accès : octobre 1, 2025, https://anjireddy-kata.medium.com/architecture-101-top-10-non-functional-requirements-nfrs-you-should-be-aware-of-c6e874bd57e0
69. Non-Functional Requirements: Tips, Tools, and Examples - Perforce Software, dernier accès : octobre 1, 2025, https://www.perforce.com/blog/alm/what-are-non-functional-requirements-examples