Plateforme d'Entreprise Agentique : Dossier d'Architecture Technique (DAT)

(Version 1.0)

1. Introduction

Ce Dossier d'Architecture Technique (DAT) constitue le document de référence fondateur pour la conception, le développement, le déploiement et l'exploitation de la Plateforme d'Entreprise Agentique. Il a pour vocation de consolider la vision technique, de justifier les décisions architecturales structurantes et de servir de guide immuable pour toutes les parties prenantes impliquées dans le cycle de vie du système. Ce document articule non seulement le « comment » de l'architecture, mais surtout le « pourquoi » des choix qui ont été faits, en liant systématiquement les exigences métier aux solutions techniques mises en œuvre.

1.1. Mission et Vision du Projet

Mission

La mission fondamentale de la Plateforme d'Entreprise Agentique est de doter l'organisation d'une capacité d'automatisation de nouvelle génération pour ses processus métiers les plus complexes. Il s'agit de dépasser l'automatisation de tâches simples pour orchestrer, de bout en bout, des flux de travail impliquant de multiples systèmes et décisions. Pour ce faire, la plateforme s'appuie sur le concept d'IA agentique : un ensemble d'agents logiciels autonomes et spécialisés, agissant de manière proactive pour atteindre des objectifs métier prédéfinis avec une intervention humaine minimale.[1]

Un agent est un système logiciel capable de percevoir son environnement, de raisonner sur les informations collectées et d'agir de manière autonome pour accomplir des tâches

spécifiques.[2] Par exemple, là où une automatisation traditionnelle se contente de générer un e-mail de suivi, un système agentique peut décider du moment opportun pour l'envoyer, personnaliser son contenu en fonction de l'historique du client, l'envoyer via l'API du service de messagerie, et mettre à jour le CRM en conséquence, le tout sans supervision directe.[3]

La mission de la plateforme est donc de fournir le socle technique pour :

1. **Modéliser** explicitement les processus métier complexes.
2. **Orchestrer** l'exécution coordonnée d'une flotte d'agents intelligents.
3. **Garantir** la fiabilité, la traçabilité et la gouvernance de ces exécutions automatisées.

Vision

La vision à long terme est de transformer la Plateforme d'Entreprise Agentique en système nerveux central des opérations de l'entreprise. Cette vision repose sur l'idée de créer une organisation "composable", où les capacités métier sont encapsulées dans des agents réutilisables. De nouveaux processus et produits pourront ainsi être assemblés et déployés avec une agilité sans précédent, en orchestrant des agents existants et nouveaux.[4]

Cette plateforme doit permettre à l'entreprise de devenir intrinsèquement plus adaptative et résiliente. En décomposant les problèmes complexes en tâches agentiques, elle accélérera l'innovation et permettra une réaction en temps réel aux dynamiques du marché.[6] La vision est celle d'une entreprise où l'automatisation n'est plus un simple outil d'efficacité pour les tâches existantes, mais un moteur stratégique pour la création de nouvelles formes de valeur, tout en assurant une gouvernance et une auditabilité complètes des processus les plus critiques.[7]

1.2. Portée de l'Architecture

Ce document définit l'architecture de la plateforme socle.

Périmètre Inclus :
Le périmètre de cette architecture englobe l'ensemble des composants, patrons et standards nécessaires à l'orchestration des agents. Sont inclus 9 :

- **Le moteur d'orchestration de processus** : Le cœur qui exécute les modèles de processus et maintient leur état.
- **L'infrastructure de communication événementielle** : Le système de messagerie qui permet une communication asynchrone et découplée.

- **La plateforme d'exécution des agents** : L'environnement (cluster de conteneurs) où les agents sont déployés, exécutés et mis à l'échelle.
- **Les mécanismes transversaux** : Les stratégies et outils pour la gouvernance, la sécurité, l'observabilité, la gestion des données et la résilience.
- **Les standards de développement** : Les cadres (SDK), conventions et modèles pour la création et le déploiement des processus et des agents.

Périmètre Exclu :
Ce document ne prétend pas décrire les éléments suivants, qui sont considérés comme externes à l'architecture de la plateforme elle-même :

- **La logique métier interne des agents** : L'implémentation spécifique de chaque agent (par exemple, les algorithmes de machine learning, les règles de validation métier) est de la responsabilité des équipes de développement et est considérée comme une "boîte noire" par la plateforme.
- **L'architecture des systèmes tiers** : Les systèmes externes (applications legacy, services SaaS, bases de données partenaires) avec lesquels les agents interagissent sont hors du périmètre de ce document. La plateforme définit uniquement les interfaces et les contrats pour ces interactions.
- **Les processus organisationnels** : Les méthodologies de gestion de projet (ex: Agile, Scrum) ou les processus de gouvernance d'entreprise ne sont pas décrits ici, bien que l'architecture soit conçue pour les supporter et s'y intégrer harmonieusement.[11]

1.3. Public Cible du Document

Ce document a été rédigé pour servir de référence à une audience variée, chaque groupe y trouvant les informations pertinentes pour ses responsabilités.[12]

- **Architectes d'Entreprise et de Solutions** : Pour valider l'alignement de la plateforme avec la stratégie globale du système d'information, comprendre les patrons directeurs et intégrer de futurs projets.
- **Développeurs et Ingénieurs DevOps** : Comme guide technique pour l'implémentation des processus et des agents, la configuration des pipelines CI/CD et le déploiement de l'infrastructure via l'Infrastructure-as-Code.
- **Chefs de Projet et Product Owners** : Pour appréhender les capacités fonctionnelles et non fonctionnelles, les contraintes et les limites de la plateforme afin de planifier efficacement les feuilles de route.
- **Équipes de Sécurité et de Conformité** : Pour auditer les contrôles de sécurité, les mécanismes de chiffrement, la gestion des accès et les capacités de traçabilité garantissant la conformité réglementaire.
- **Équipes d'Exploitation (Opérations)** : Pour comprendre l'architecture de production,

les stratégies de surveillance, les plans de haute disponibilité et les procédures de reprise après sinistre.

1.4. Objectifs Architecturaux Clés (Attributs de Qualité)

Les attributs de qualité, ou exigences non fonctionnelles, sont les principaux moteurs des décisions architecturales. Ils définissent les caractéristiques opérationnelles et structurelles que le système doit posséder pour répondre aux attentes métier.

1.4.1. Gouvernance et Auditabilité

- **Exigence** : Chaque exécution de processus métier doit être entièrement traçable. Il doit être possible de reconstituer la séquence exacte des événements, de savoir quel agent a exécuté quelle tâche, à quel moment, avec quelles données d'entrée et de sortie. L'état de chaque instance de processus (en cours, terminée, en erreur) doit être visible en temps réel.
- **Justification** : Cet attribut est non négociable pour la conformité réglementaire (ex: GDPR, SOX), la résolution d'incidents, l'analyse des causes racines et l'optimisation continue des processus.[7]
- **Mécanismes** : L'utilisation d'un moteur d'orchestration explicite comme Camunda 8, qui persiste un historique détaillé de chaque événement de processus, est fondamentale. Ceci est complété par une journalisation centralisée des actions des agents et l'utilisation d'identifiants de corrélation uniques qui lient les journaux d'application à une instance de processus spécifique.

1.4.2. Évolutivité et Scalabilité

- **Exigence** : La plateforme doit pouvoir s'adapter de manière élastique aux variations de charge. Cela inclut la capacité à gérer une augmentation du nombre de processus démarrés par seconde (scalabilité du moteur) et une augmentation du nombre de tâches exécutées en parallèle (scalabilité des agents), sans dégradation perceptible des performances.
- **Justification** : La plateforme doit supporter la croissance de l'entreprise, les pics d'activité (ex: soldes, fin de trimestre fiscal) et l'intégration de nouveaux processus sans

nécessiter de refonte architecturale.[17]

- **Mécanismes** : Une architecture nativement conçue pour le cloud et déployée sur Kubernetes. Le moteur Zeebe de Camunda 8 est conçu pour une scalabilité horizontale en ajoutant des nœuds au cluster. Les agents, en tant que microservices indépendants, peuvent être mis à l'échelle horizontalement via les mécanismes de Kubernetes (Horizontal Pod Autoscaler). L'utilisation d'une plateforme de streaming managée comme Confluent Cloud garantit que le bus de communication peut également absorber des charges élevées.

1.4.3. Résilience et Fiabilité

- **Exigence** : Le système doit être tolérant aux pannes. La défaillance d'un agent individuel, d'un nœud de calcul ou d'un composant d'infrastructure ne doit pas entraîner la perte de l'état d'un processus en cours. Le système doit pouvoir reprendre son exécution de manière cohérente après la résolution de la panne.
- **Justification** : Les processus orchestrés sont critiques. Leur interruption ou la perte de leur état peut avoir des conséquences financières et réputationnelles directes pour l'entreprise.[19]
- **Mécanismes** : L'état de chaque processus est persisté de manière durable par le moteur d'orchestration. Le patron *External Task Worker* permet au moteur de re-proposer une tâche si un agent ne la termine pas dans le temps imparti. Des mécanismes de nouvelles tentatives (retries) sont configurables au niveau de chaque tâche. Le patron Saga est utilisé pour gérer les erreurs métier et garantir la cohérence transactionnelle. Enfin, les composants de la plateforme sont déployés de manière redondante sur plusieurs zones de disponibilité.

1.4.4. Maintenabilité et Interopérabilité

- **Exigence** : Il doit être possible de développer, tester, déployer, mettre à jour et retirer un agent de manière indépendante, sans affecter le reste de la plateforme. Différentes équipes doivent pouvoir travailler en parallèle sur différents agents, en utilisant les technologies les plus appropriées pour chaque tâche.
- **Justification** : La maintenabilité est cruciale pour réduire le coût total de possession (TCO) et accélérer le "time-to-market". L'interopérabilité permet de capitaliser sur une diversité de compétences et d'outils, évitant le verrouillage technologique.[17]
- **Mécanismes** : Le découplage fort entre l'orchestrateur et les agents, réalisé grâce au patron *External Task Worker*, est le mécanisme central. La communication asynchrone

par événements renforce ce découplage. L'utilisation de conteneurs (Docker) et d'API standardisées (REST, gRPC) garantit l'interopérabilité technologique.

Le tableau suivant établit une cartographie explicite entre ces objectifs et les mécanismes architecturaux choisis pour les atteindre, démontrant ainsi que l'architecture est une réponse délibérée et réfléchie aux exigences non fonctionnelles.

Tableau 1 : Mapping des Attributs de Qualité aux Mécanismes Architecturaux

Attribut de Qualité	Mécanismes Architecturaux Clés
Gouvernance et Auditabilité	Orchestration de processus explicite (BPMN), Moteur Camunda 8 (historique d'exécution), Journalisation centralisée, Identifiants de corrélation.
Évolutivité et Scalabilité	Déploiement sur Kubernetes (HPA), Moteur Zeebe (scalabilité horizontale), Agents en tant que microservices, Confluent Cloud (scalabilité managée).
Résilience et Fiabilité	Persistance de l'état du processus, Patron External Task Worker (avec verrouillage et timeout), Patron Saga (compensation), Déploiement multi-AZ.
Maintenabilité et Interopérabilité	Découplage fort (Processus/Exécution), Patron External Task Worker, Communication asynchrone par événements, Conteneurisation (Docker), API standardisées.

1.5. Contraintes Architecturales (Techniques, Organisationnelles, Budgétaires)

L'architecture n'est pas conçue dans le vide ; elle est façonnée par un ensemble de contraintes qui limitent l'espace des solutions possibles.

- **Contraintes Techniques** :
  - **Écosystème Existant** : La plateforme doit s'intégrer avec le fournisseur d'identité de l'entreprise (IdP) pour l'authentification et la gestion des autorisations (SSO).
  - **Plateforme Cible** : Le déploiement doit s'effectuer sur l'infrastructure Kubernetes gérée par l'entreprise, en respectant ses standards de sécurité et d'exploitation.
  - **Préférence pour le Managé** : Afin de réduire la charge opérationnelle et de se concentrer sur la valeur métier, une préférence est accordée aux services cloud managés pour les dépendances critiques (ex: Confluent Cloud pour Kafka, bases de données managées pour le stockage).[22]
- **Contraintes Organisationnelles** :
  - **Modèle Opérationnel** : L'architecture doit être compatible avec un modèle organisationnel basé sur des équipes produits autonomes et décentralisées, responsables de leurs agents de bout en bout ("You build it, you run it").
  - **Compétences** : La conception doit tenir compte des compétences existantes et prévoir un plan de montée en compétences pour les technologies clés (BPMN, Kubernetes, Kafka, Go/Java).
- **Contraintes Budgétaires** :
  - **Optimisation des Coûts** : Les coûts de licence et d'infrastructure doivent être maîtrisés et prévisibles. Le choix d'une pile technologique basée sur un cœur open-source (Kubernetes, BPMN) complété par des solutions commerciales pour les aspects critiques (Camunda pour l'orchestration, Confluent pour le streaming) représente un compromis stratégique entre coût, support, fonctionnalités avancées et réduction des risques.

2. Principes et Décisions d'Architecture

Cette section constitue le fondement logique de l'architecture. Elle énonce les principes directeurs qui ont guidé chaque décision et formalise les choix les plus structurants dans un registre immuable. Ces principes ne sont pas de simples recommandations ; ce sont des règles contraignantes qui garantissent la cohérence et l'intégrité de la plateforme à long terme.

2.1. Principes Directeurs de l'Architecture

2.1.1. Modélisation Explicite des Processus (BPMN-First)

- **Principe** : Tout processus métier orchestré par la plateforme doit être formalisé à l'aide de la notation standard *Business Process Model and Notation* (BPMN) 2.0. Ce modèle BPMN n'est pas une simple documentation a posteriori ; il est la source unique de vérité (single source of truth) et est directement interprétable et exécutable par le moteur de processus.[24]
- **Conséquences** :
  - **Langage Commun** : Le BPMN établit un pont entre les parties prenantes métier et techniques. Les analystes métier peuvent concevoir et valider le flux logique du processus, tandis que les développeurs l'implémentent en sachant que le modèle graphique est une représentation fidèle de ce qui sera exécuté.[24]
  - **Gouvernance Intégrée** : La logique de contrôle (les branchements conditionnels, le parallélisme, la gestion des erreurs) est visible et explicite dans le diagramme. Cela rend le processus auditable par nature et facilite grandement l'analyse des performances et l'identification des goulots d'étranglement.[25]
  - **Agilité et Évolution** : Modifier le déroulement d'un processus (ajouter une étape, changer une condition) se fait en modifiant le modèle BPMN, puis en le redéployant. Cette approche est nettement plus agile et moins risquée que de devoir modifier une logique de contrôle complexe et implicite, disséminée dans du code impératif.

2.1.2. Communication Asynchrone et Événementielle par Défaut

- **Principe** : Les interactions entre les composants découplés de la plateforme, et en particulier entre le moteur d'orchestration et les agents, doivent par défaut utiliser un mode de communication asynchrone basé sur des événements. Les appels synchrones bloquants sont considérés comme une exception et doivent être rigoureusement justifiés, par exemple pour des lectures de données critiques nécessitant une cohérence immédiate.
- **Conséquences** :
  - **Résilience Accrue** : Si un agent est temporairement indisponible (panne, redéploiement), le moteur d'orchestration n'est pas bloqué. Les tâches qui lui sont destinées restent en attente et seront traitées dès que l'agent sera de nouveau opérationnel. Cela permet une dégradation gracieuse du service plutôt qu'une défaillance en cascade.[17]
  - **Élasticité et Scalabilité** : Les producteurs d'événements (comme le moteur) et les consommateurs (les agents) peuvent être mis à l'échelle indépendamment. Si une file d'attente de tâches s'allonge, seul le groupe d'agents concerné doit être mis à l'échelle, sans impacter les autres composants du système.[29]

    - **Découplage Temporel** : Les composants n'ont pas besoin d'être actifs simultanément pour communiquer. Un processus peut initier une tâche dont l'exécution par un agent peut avoir lieu des secondes, minutes ou même heures plus tard.

2.1.3. Découplage Fort entre le Processus et l'Exécution

- **Principe** : Une séparation stricte des responsabilités est appliquée. Le moteur d'orchestration est responsable du **"quoi"** et du **"quand"** (la séquence des étapes, la logique de flux, la gestion de l'état). Les agents sont responsables du **"comment"** (l'implémentation concrète de la logique métier de chaque étape). Le moteur ne doit avoir aucune connaissance de la technologie, de la localisation ou de la logique interne des agents qu'il orchestre.
- **Conséquences** :
    - **Autonomie des Équipes et Polyglottisme** : Chaque agent peut être développé, testé et déployé par une équipe autonome en utilisant la pile technologique la plus adaptée à sa fonction (Java, Go, Python, etc.). La seule exigence est sa capacité à communiquer avec la plateforme via les API définies.[30]
    - **Maintenabilité et Évolutivité Indépendantes** : Une équipe peut refactoriser, optimiser ou même réécrire complètement un agent sans aucun impact sur le modèle de processus ou les autres agents, tant que le contrat d'interface (les variables d'entrée/sortie) est respecté.[31]
    - **Réutilisabilité des Capacités Métier** : Un agent encapsulant une capacité métier générique (ex: "Vérifier la solvabilité d'un client", "Envoyer une notification SMS") peut être conçu une seule fois et réutilisé dans de multiples processus métier, accélérant ainsi le développement de nouvelles fonctionnalités.

2.1.4. Infrastructure Immuable et Déclarative (Infrastructure-as-Code)

- **Principe** : L'intégralité de l'infrastructure de la plateforme (clusters Kubernetes, configurations réseau, politiques de sécurité, déploiements applicatifs) est définie sous forme de code à l'aide d'outils déclaratifs (Terraform, Helm). Ce code est stocké et versionné dans un dépôt Git. Les mises à jour ne consistent pas à modifier un environnement existant, mais à déployer une nouvelle version de l'infrastructure qui remplace l'ancienne.
- **Conséquences** :
    - **Reproductibilité et Cohérence** : Ce principe élimine la "dérive de configuration" (configuration drift) en garantissant que tous les environnements (développement,

test, production) sont créés à partir de la même base de code, assurant ainsi leur cohérence.[22]
    - **Auditabilité et Traçabilité** : Tout changement apporté à l'infrastructure est tracé dans l'historique Git. Les revues de code (Pull Requests) pour les modifications d'infrastructure permettent une validation par les pairs et créent une piste d'audit complète.[18]
    - **Automatisation et Fiabilité des Déploiements** : Les processus de provisionnement, de mise à jour et de reprise après sinistre sont entièrement automatisés, ce qui les rend plus rapides, plus fiables et moins sujets aux erreurs humaines que les interventions manuelles.[23]

2.2. Patrons Architecturaux Appliqués

Les principes directeurs sont mis en œuvre à travers un ensemble de patrons architecturaux éprouvés, choisis pour leur adéquation avec les objectifs de la plateforme.

2.2.1. Orchestration de Processus (Process Orchestration)

- **Description** : Ce patron implique un composant central, l'orchestrateur (ici, le moteur Camunda), qui coordonne activement les interactions entre plusieurs services participants (les agents). L'orchestrateur maintient l'état du processus de bout en bout et dicte explicitement la séquence des opérations à exécuter en envoyant des commandes aux services.[33]
- **Justification** : Pour les processus métier critiques qui constituent le cœur de la plateforme, la visibilité, la traçabilité et la capacité à gérer centralement les erreurs sont primordiales. L'orchestration rend le flux de contrôle explicite et observable, ce qui est fondamental pour l'attribut de qualité "Gouvernance et Auditabilité". Contrairement à la chorégraphie, où le flux est implicite et distribué, l'orchestration permet de savoir à tout instant où en est un processus et de gérer des logiques de compensation complexes de manière centralisée.[33]

2.2.2. Travailleur de Tâche Externe (External Task Worker)

- **Description** : Ce patron inverse le flux de communication traditionnel de l'orchestration.

Au lieu que l'orchestrateur appelle directement un service (mode "push"), il publie une unité de travail (une "tâche externe" ou "job") dans une liste. Des applications externes, les "workers" (nos agents), interrogent périodiquement le moteur (mode "pull") pour demander des tâches disponibles pour les sujets (topics) auxquels ils sont abonnés. Lorsqu'un worker récupère une tâche, il la verrouille pour une durée déterminée, exécute la logique métier, puis notifie le moteur de son achèvement ou de son échec.[30]

- **Justification** : C'est le patron architectural clé qui permet de respecter le principe de découplage fort tout en bénéficiant de l'orchestration. Il découple l'orchestrateur de l'implémentation des workers, qui peuvent être hétérogènes, évoluer indépendamment et être mis à l'échelle séparément. Il améliore également la résilience : si un worker est en panne, les tâches s'accumulent simplement dans la file d'attente du moteur jusqu'à ce qu'il redevienne disponible.[30]

2.2.3. Architecture Orientée Événements (Event-Driven Architecture)

- **Description** : L'architecture est structurée autour de la production, de la détection et de la consommation d'événements. Un événement représente un fait significatif qui s'est produit (ex: "Commande Créée"). Les composants communiquent de manière asynchrone via un intermédiaire, le bus d'événements (ici, Confluent Cloud), qui achemine les événements des producteurs vers les consommateurs intéressés.[17]
- **Justification** : Ce patron est le fondement de la communication asynchrone et du découplage. Il permet aux systèmes de réagir en temps réel aux changements d'état, favorise une scalabilité élevée et une grande résilience. Il offre également une extensibilité remarquable : de nouveaux services peuvent s'abonner à des flux d'événements existants pour implémenter de nouvelles fonctionnalités sans nécessiter de modification des services producteurs.[29]

2.2.4. Compensation Transactionnelle (Saga Pattern)

- **Description** : Dans une architecture de microservices, il est déconseillé d'utiliser des transactions distribuées (protocole 2PC) qui verrouillent les ressources sur plusieurs services. Le patron Saga gère la cohérence des données en décomposant une transaction métier globale en une séquence de transactions locales, chaque service étant responsable de sa propre transaction. Si une transaction locale échoue, la saga exécute des transactions de compensation pour annuler les effets des transactions locales précédentes qui avaient réussi.[40]
- **Justification** : C'est le patron standard pour garantir la cohérence des données au

niveau métier dans un système distribué. Il préserve l'autonomie de chaque service (pas de verrous partagés) et évite les goulots d'étranglement de performance associés aux transactions distribuées, tout en assurant que le système peut être ramené à un état cohérent en cas d'erreur.[40] L'orchestrateur de processus est l'endroit idéal pour implémenter la logique de la saga, car il suit naturellement l'état de la transaction globale.

2.3. Registre des Décisions d'Architecture (ADR - Architecture Decision Records)

Cette section documente les décisions architecturales les plus importantes de manière formelle. Chaque ADR suit une structure standard : Contexte, Décision, et Justification/Conséquences.

2.3.1. ADR-001 : Choix du Modèle d'Orchestration (vs. Chorégraphie)

- **Statut** : Accepté.
- **Contexte** : La plateforme doit coordonner les actions de multiples agents pour exécuter des processus métier complexes. Deux patrons dominants existent : l'orchestration, avec un contrôle centralisé, et la chorégraphie, avec un contrôle décentralisé et réactif aux événements.
- **Décision** : Adopter le modèle d'**Orchestration** comme patron principal pour les processus métier structurés et critiques. La chorégraphie pourra être considérée pour des cas d'usage spécifiques et moins critiques dans les évolutions futures de la plateforme.
- **Justification** :
  1. **Visibilité et Auditabilité** : L'orchestration offre une vue centralisée et explicite du flux de processus via le modèle BPMN. C'est un avantage décisif pour répondre à notre objectif de gouvernance. Avec la chorégraphie, le flux global est implicite, réparti entre les services, ce qui rend le suivi et le débogage d'une transaction de bout en bout extrêmement complexes.[33]
  2. **Gestion des Erreurs** : La gestion des erreurs métier, des délais (timeouts) et des logiques de compensation (Saga) est considérablement simplifiée lorsqu'elle est gérée par un orchestrateur central qui a une connaissance complète de l'état du processus.[33]
  3. **Complexité du Workflow** : Pour les processus comportant de nombreuses étapes, des dépendances conditionnelles et des boucles, un orchestrateur central réduit la complexité de la logique qui devrait sinon être implémentée dans chaque service

participant.[35]
    - **Conséquences** : Le moteur d'orchestration devient un composant critique du système. Sa haute disponibilité doit être assurée. Il existe un risque perçu de couplage plus fort, mais ce risque est atténué par l'utilisation systématique du patron *External Task Worker*.

2.3.2. ADR-002 : Choix de Camunda 8 comme Moteur d'Orchestration

- **Statut** : Accepté.
- **Contexte** : Le choix d'un modèle d'orchestration impose la sélection d'un moteur capable d'exécuter des modèles de processus, de gérer l'état de millions d'instances, et de s'intégrer de manière flexible avec un écosystème de services externes.
- **Décision** : Sélectionner **Camunda Platform 8** comme moteur d'orchestration de la plateforme.
- **Justification** :
    1. **Alignement Architectural** : Camunda 8, avec son moteur Zeebe, est conçu comme une plateforme cloud-native. Son architecture sans base de données externe pour la gestion de l'état, basée sur un log d'événements répliqué, est optimisée pour la scalabilité horizontale et la résilience, ce qui s'aligne parfaitement avec notre déploiement cible sur Kubernetes.[46]
    2. **Standardisation** : Camunda 8 utilise et exécute nativement le standard BPMN 2.0, ce qui est en parfaite adéquation avec notre principe "BPMN-First".[46]
    3. **Découplage** : La plateforme supporte nativement et promeut le patron *External Task Worker*, qui est fondamental pour notre principe de découplage fort entre le processus et l'exécution.[48]
    4. **Écosystème Complet** : Camunda 8 fournit une suite d'outils intégrés (Operate pour la supervision, Tasklist pour les tâches humaines, Web Modeler pour la conception collaborative) qui accélèrent le développement et l'exploitation.[49]
    - **Conséquences** : L'adoption de Camunda 8 implique un investissement dans une technologie spécifique et la montée en compétences des équipes sur son écosystème.

2.3.3. ADR-003 : Choix de Confluent Cloud comme Plateforme de Streaming d'Événements

- **Statut** : Accepté.
- **Contexte** : Une communication asynchrone et événementielle fiable nécessite une

infrastructure de messagerie robuste, scalable, et hautement disponible. Le choix se situe entre une solution auto-hébergée (ex: RabbitMQ, Apache Kafka) et une solution managée.

- **Décision** : Utiliser **Confluent Cloud** comme plateforme de streaming d'événements.
- **Justification** :
  1. **Réduction de la Charge Opérationnelle** : Confluent Cloud est une offre "Kafka-as-a-Service" entièrement managée. Cela nous décharge de la complexité opérationnelle considérable liée à la gestion, la sécurisation, la mise à l'échelle et la maintenance d'un cluster Kafka en production.[50]
  2. **Performance et Scalabilité** : Apache Kafka, le cœur de Confluent, est la technologie de référence pour le streaming d'événements à haut débit et faible latence, capable de supporter des millions d'événements par seconde.[52]
  3. **Gouvernance et Écosystème** : Confluent Cloud inclut des outils essentiels pour une architecture événementielle mature, notamment le *Schema Registry* pour la gouvernance des schémas d'événements (évitant les "breaking changes") et un large catalogue de connecteurs pour s'intégrer facilement avec d'autres systèmes.[53]
  - **Conséquences** : Dépendance envers un fournisseur tiers et coûts associés au service managé, qui doivent être suivis et optimisés.

2.3.4. ADR-004 : Choix de Kubernetes comme Plateforme d'Exécution des Agents

- **Statut** : Accepté.
- **Contexte** : Les agents, conçus comme des microservices, nécessitent une plateforme pour leur déploiement, leur gestion de cycle de vie, leur mise à l'échelle et leur résilience.
- **Décision** : Utiliser **Kubernetes** comme plateforme d'orchestration de conteneurs pour l'exécution des agents et des composants de la plateforme.
- **Justification** :
  1. **Standard de l'Industrie** : Kubernetes est devenu la norme de facto pour l'orchestration de conteneurs, bénéficiant d'un écosystème immense et d'un large soutien de la communauté et des fournisseurs de cloud.[55]
  2. **Alignement avec les Attributs de Qualité** : Kubernetes fournit nativement les mécanismes nécessaires pour atteindre nos objectifs de scalabilité (Horizontal Pod Autoscaler), de résilience (auto-réparation des pods, déploiements progressifs) et de maintenabilité (déploiements déclaratifs).[57]
  3. **Cohérence de la Pile Technologique** : Camunda 8 et Confluent Cloud (via des opérateurs Kubernetes) sont conçus pour être déployés et gérés de manière native sur Kubernetes. Le choix de Kubernetes crée une plateforme d'exécution homogène et cohérente pour tous les composants de l'architecture.[59]
  - **Conséquences** : Kubernetes introduit une courbe d'apprentissage et une complexité opérationnelle. Cependant, cette complexité est un investissement justifié par les

gains en automatisation, en résilience et en scalabilité à long terme.

La cohérence de ces décisions est un point fondamental. Il ne s'agit pas d'une simple juxtaposition de technologies leaders, mais de la construction d'un système où chaque composant renforce les autres. Kubernetes offre un environnement d'exécution résilient et scalable, ce qui est une condition sine qua non pour une plateforme cloud-native. Le moteur Zeebe de Camunda 8 est spécifiquement conçu pour tirer parti de cet environnement : son architecture de log répliqué, qui évite la dépendance à une base de données relationnelle traditionnelle pour l'état d'exécution, est une incarnation du design pour la scalabilité et la tolérance aux pannes que Kubernetes promeut. De même, Apache Kafka (le cœur de Confluent) est lui-même un système distribué qui s'épanouit dans un environnement comme Kubernetes. L'ensemble forme une architecture synergique, alignée sur les principes du cloud-natif de bout en bout, ce qui maximise les bénéfices de chaque composant et réduit les frictions d'intégration.

3. Vues Architecturales (Modèle 4+1)

Pour décrire cette architecture complexe de manière exhaustive et compréhensible par les différentes parties prenantes, ce document adopte le modèle de vue architecturale 4+1 de Philippe Kruchten.[15] Ce modèle organise la description de l'architecture en quatre vues interdépendantes (Logique, Processus, Développement, Déploiement), qui sont validées et illustrées par une cinquième vue transversale (Cas d'Usage).

3.1. Vue Logique : Décomposition Fonctionnelle

Cette vue se concentre sur la décomposition fonctionnelle du système et ses responsabilités, du point de vue des services qu'il offre. Elle décrit les abstractions et les domaines fonctionnels majeurs, indépendamment de leur implémentation technique ou de leur déploiement physique.[64]

3.1.1. Domaine de l'Orchestration de Processus

- **Responsabilité** : Ce domaine est le cœur de la plateforme. Il est responsable de la modélisation, de l'exécution, de la persistance de l'état et de la gestion du cycle de vie

des processus métier.
- **Composants Logiques** :
  - **Moteur de Workflow** : L'entité qui interprète les modèles BPMN, crée des instances de processus, gère leur état et planifie l'exécution des tâches.
  - **Modélisateur de Processus** : L'outil (ici, web) qui permet aux utilisateurs de concevoir, de commenter et de déployer les modèles de processus BPMN et les tables de décision DMN.
  - **API de Processus** : L'interface de programmation qui permet aux applications externes de démarrer des instances de processus, d'envoyer des messages, de consulter leur état, etc.

3.1.2. Domaine de la Communication Événementielle

- **Responsabilité** : Assurer le transport fiable, ordonné, durable et asynchrone des messages et des événements entre tous les composants découplés de la plateforme.
- **Composants Logiques** :
  - **Bus d'Événements (Event Broker)** : Le canal central par lequel les événements sont publiés et consommés. Il garantit la persistance des messages.
  - **Registre de Schémas (Schema Registry)** : Un service qui stocke et valide les schémas des événements, garantissant que les producteurs et les consommateurs s'accordent sur la structure des données échangées.
  - **Connecteurs** : Composants spécialisés pour l'intégration avec des systèmes externes, en source (pour ingérer des événements) ou en destination (pour propager des événements).

3.1.3. Domaine de l'Exécution Agentique

- **Responsabilité** : Exécuter la logique métier concrète de chaque étape d'un processus. C'est ici que le "travail" est réellement effectué.
- **Composants Logiques** :
  - **Bibliothèque Client pour Worker (SDK)** : Une bibliothèque fournie aux développeurs d'agents pour simplifier l'interaction avec le moteur d'orchestration (récupération des tâches, complétion, gestion des erreurs).
  - **Agent Spécifique** : L'implémentation d'une capacité métier atomique (ex: AgentDeValidationCredit, AgentDeGenerationPDF, AgentDeNotificationClient). Chaque agent est un consommateur de tâches et un producteur de résultats.

3.1.4. Domaine de la Gouvernance et de l'Observabilité

- **Responsabilité** : Fournir une visibilité complète sur la santé et la performance du système, permettre l'intervention humaine lorsque nécessaire, et collecter les données pour l'audit et l'amélioration.
- **Composants Logiques** :
  - **Outil de Supervision et de Dépannage (Monitoring Tool)** : Une interface utilisateur pour visualiser les instances de processus en temps réel, inspecter leur état, analyser les erreurs et résoudre les incidents.
  - **Gestionnaire de Tâches Humaines (Human Task Manager)** : Une application (liste de tâches) où les utilisateurs humains peuvent réclamer et compléter des tâches qui leur sont assignées dans un processus.
  - **Système d'Observabilité** : Un ensemble de services pour la collecte, l'agrégation et la visualisation des **journaux (logs)**, des **métriques (metrics)** et des **traces distribuées (traces)** de tous les composants de la plateforme.

3.2. Vue des Processus : Interactions et Comportement d'Exécution

Cette vue décrit les aspects dynamiques et le comportement du système à l'exécution. Elle illustre comment les composants logiques interagissent à travers des protocoles de communication pour réaliser des cas d'usage spécifiques. Des diagrammes de séquence sont utilisés pour clarifier ces interactions complexes.[62]

3.2.1. Séquence de Démarrage d'un Processus

Ce diagramme illustre la création d'une nouvelle instance de processus.

1. **Client API** -> **Gateway Zeebe** : Envoie une requête CreateProcessInstance (gRPC) avec l'ID du processus et les variables initiales.
2. **Gateway Zeebe** -> **Broker Zeebe (Leader)** : Transmet la commande au broker leader de la partition appropriée.
3. **Broker Zeebe** : Valide la commande, écrit un événement PROCESS_INSTANCE_CREATED dans son log interne, le réplique sur les brokers suiveurs pour la durabilité.
4. **Broker Zeebe** -> **Gateway Zeebe** : Confirme la création de l'instance.

5. **Gateway Zeebe** -> **Client API** : Retourne une réponse de succès avec l'identifiant de la nouvelle instance de processus.
6. **Broker Zeebe** : Active la première tâche du processus. Si c'est une tâche de service externe, il écrit un événement JOB_CREATED dans le log, rendant la tâche disponible pour les workers.

3.2.2. Séquence d'Exécution d'une Tâche par un Agent

Ce diagramme montre le cycle de vie complet d'une tâche externe.

1. **Agent Worker** -> **Gateway Zeebe** : Envoie une requête ActivateJobs (gRPC, long-polling) pour un jobType spécifique (ex: "validation-credit"). La requête reste ouverte si aucune tâche n'est disponible.
2. **Gateway Zeebe** -> **Brokers Zeebe** : Diffuse la demande d'activation aux brokers.
3. **Broker Zeebe** : Lorsqu'une tâche correspondante devient disponible, il la verrouille pour le worker demandeur (en définissant un propriétaire et un deadline), écrit un événement JOB_ACTIVATED et envoie la tâche (avec ses variables) au Gateway.
4. **Gateway Zeebe** -> **Agent Worker** : Transmet la tâche activée.
5. **Agent Worker** : Reçoit la tâche et exécute sa logique métier (ex: appel à une API de score de crédit).
6. **Agent Worker** -> **Gateway Zeebe** : Une fois le travail terminé, envoie une requête CompleteJob avec l'identifiant de la tâche et les variables de résultat.
7. **Gateway Zeebe** -> **Broker Zeebe (Leader)** : Transmet la commande de complétion.
8. **Broker Zeebe** : Valide que le worker détient bien le verrou, écrit un événement JOB_COMPLETED, met à jour l'état de l'instance de processus, et active la tâche suivante dans le workflow.

3.2.3. Séquence de Gestion d'une Erreur et Compensation (Saga)

Ce diagramme illustre comment le patron Saga est orchestré en cas d'erreur métier.

1. **Agent Worker (Paiement)** : Tente d'exécuter une tâche de paiement mais reçoit une réponse "Fonds insuffisants" de la passerelle de paiement. Il s'agit d'une erreur métier prévisible.
2. **Agent Worker (Paiement)** -> **Gateway Zeebe** : Au lieu de compléter la tâche, il envoie une commande ThrowError avec un code d'erreur spécifique (ex: "ERR_SOLDE_INSUFFISANT").
3. **Gateway Zeebe** -> **Broker Zeebe (Leader)** : Transmet la commande d'erreur.

4. **Broker Zeebe** : Intercepte l'erreur. Au lieu de suivre le chemin nominal, il consulte le modèle BPMN et trouve un événement de frontière d'erreur (error boundary event) attaché à la tâche de paiement qui correspond à ce code d'erreur.
5. **Broker Zeebe** : Active le flux d'exception défini après cet événement. Ce flux mène à une tâche de compensation (ex: "Annuler la réservation de stock"). Un événement JOB_CREATED est émis pour cette nouvelle tâche.
6. **Agent Worker (Stock)** : Récupère et exécute la tâche "Annuler la réservation de stock", effectuant ainsi la transaction de compensation.
7. **Agent Worker (Stock)** -> **Gateway Zeebe** : Envoie CompleteJob pour la tâche de compensation.
8. **Broker Zeebe** : Le processus se termine dans un état final d'échec métier géré et cohérent.

3.3. Vue de Développement : Organisation du Code Source

Cette vue s'adresse aux développeurs et aux responsables techniques. Elle décrit comment le code source est organisé, géré et standardisé pour assurer la qualité, la cohérence et la maintenabilité.[69]

3.3.1. Structure des Dépôts de Code (Repositories)

L'architecture adopte une approche **multi-dépôts (multi-repo)**, où chaque composant ou domaine logique est géré dans son propre dépôt Git. Cette approche favorise l'autonomie des équipes et des cycles de vie de déploiement indépendants.

- **Dépôt platform-processes** : Ce dépôt centralise tous les artefacts de processus métier (fichiers .bpmn) et de décision (.dmn). Il est considéré comme un actif partagé, et toute modification suit un processus de revue rigoureux impliquant à la fois les analystes métier et les architectes.
- **Dépôt agent-<nom-agent>** : Chaque agent (ou groupe d'agents étroitement liés) possède son propre dépôt. Cela inclut le code source de l'agent, ses tests, son Dockerfile, et sa configuration de déploiement Kubernetes (Helm chart). Par exemple : agent-notification, agent-facturation.
- **Dépôt platform-infrastructure** : Ce dépôt contient tout le code d'Infrastructure-as-Code (IaC) pour provisionner et configurer la plateforme elle-même (le cluster Kubernetes, les bases de données, les configurations de Confluent Cloud, etc.) en utilisant Terraform.

- **Dépôt platform-agent-sdk** : Dépôt pour la bibliothèque partagée qui facilite le développement des agents.

3.3.2. Gestion des Dépendances et des Bibliothèques Partagées

Pour éviter la duplication de code et standardiser les interactions avec la plateforme, une bibliothèque partagée (agent-sdk) est développée.

- **Contenu du SDK** : Il abstrait la complexité de la communication avec le moteur Zeebe (via gRPC), en fournissant des méthodes simples pour s'abonner à un topic, gérer le cycle de vie des tâches (complétion, échec, extension de verrou), et manipuler les variables. Il inclut également des modules pour la journalisation structurée et l'exposition de métriques standardisées.
- **Distribution** : Le SDK est versionné et publié dans un registre d'artefacts privé (ex: Nexus, GitHub Packages). Chaque agent le consomme comme une dépendance externe standard, permettant des mises à jour centralisées et contrôlées de la logique de communication.

3.3.3. Standards de Codage et Conventions

La qualité et la cohérence du code sont assurées par l'application systématique de standards.

- **Conventions de Codage** : Chaque langage de programmation utilisé pour les agents doit adhérer à une convention de style reconnue (ex: Google Style Guides pour Java/Go, PEP 8 pour Python).
- **Qualité du Code** : L'intégration continue (CI) inclut des étapes obligatoires d'analyse de code statique (linting), de vérification de la couverture de test, et d'analyse des vulnérabilités des dépendances.
- **Revue de Code** : Aucune modification de code n'est fusionnée dans la branche principale sans une revue par les pairs via une Pull Request. Ce processus garantit le partage des connaissances, l'amélioration de la qualité et le respect des standards.[72]

3.4. Vue de Déploiement (Physique) : Topologie de l'Infrastructure

Cette vue décrit la topologie physique du système, en cartographiant les composants logiciels

sur l'infrastructure matérielle (ou virtualisée) et en détaillant les flux de communication réseau.[55]

3.4.1. Cartographie des Environnements (DEV, QA, PROD)

La plateforme est déployée sur trois environnements distincts et isolés pour garantir un cycle de vie de développement et de livraison sécurisé et fiable.

Tableau 2 : Cartographie des Environnements

Environnement	Cluster Kubernetes	Dépendances (Base de données, Kafka)	Objectif
DEV	Cluster partagé, isolation par namespace	Instances partagées, taille réduite	Développement et tests unitaires/intégration par les équipes
QA	Cluster dédié, répliquant la topologie PROD	Instances dédiées, taille réduite	Tests d'intégration de bout en bout, tests de performance, validation métier
PROD	Cluster dédié, haute disponibilité (multi-AZ)	Instances dédiées, haute disponibilité, dimensionnées pour la charge	Exécution des processus métier de production

3.4.2. Topologie de Déploiement sur Kubernetes

- **Cluster d'Orchestration Camunda 8** : Déployé via le Helm Chart officiel.
  - **Zeebe Brokers** : Déployés en tant que StatefulSet avec 3 répliques (ou plus) réparties sur différentes zones de disponibilité (AZ) pour la haute disponibilité.

Chaque pod dispose d'un PersistentVolume pour le stockage de son log.
    - **Zeebe Gateway** : Déployé en tant que Deployment avec plusieurs répliques derrière un Service de type LoadBalancer pour distribuer les requêtes des clients et des workers.
    - **Operate, Tasklist, Identity** : Déployés en tant que Deployments standards.
- **Agents (Workers)** :
    - Chaque type d'agent est packagé dans une image Docker et déployé en tant que Deployment Kubernetes.
    - Un HorizontalPodAutoscaler (HPA) est configuré pour chaque déploiement d'agent, basé sur des métriques comme l'utilisation du CPU ou des métriques personnalisées (ex: la profondeur de la file d'attente des tâches).
- **Dépendances Externes** :
    - Les bases de données (Elasticsearch pour Operate/Tasklist, PostgreSQL pour Identity) sont provisionnées en tant que services managés (ex: AWS OpenSearch, AWS RDS) en dehors du cluster Kubernetes. Cela sépare le cycle de vie des données de celui des applications et tire parti de la résilience et de la gestion offertes par le fournisseur de cloud.[59]

3.4.3. Topologie Réseau et Flux de Communication Sécurisés

- **Trafic Entrant (Ingress)** : Un Ingress Controller (ex: NGINX Ingress) est déployé pour gérer tout le trafic HTTP/S entrant. Il expose les interfaces utilisateur (Operate, Tasklist) et les API REST. Le trafic gRPC vers le Gateway Zeebe est géré via un Service de type LoadBalancer.
- **Communication Interne** : La communication entre les pods au sein du cluster est restreinte par défaut. Des Network Policies sont définies pour n'autoriser que les flux de communication explicitement nécessaires (ex: autoriser les agents à communiquer avec le Gateway Zeebe, mais interdire la communication directe entre agents).
- **Trafic Sortant (Egress)** : Tout le trafic sortant du cluster vers des services externes (Confluent Cloud, API partenaires) est routé à travers une passerelle NAT pour contrôler et sécuriser les flux. Les communications sont systématiquement chiffrées via TLS.

3.5. Vue des Cas d'Usage : Validation de l'Architecture

Cette vue, le "+1" du modèle, utilise des scénarios concrets pour traverser l'architecture et démontrer comment les différentes vues collaborent pour répondre aux exigences

fonctionnelles et non fonctionnelles.[15]

3.5.1. Scénario : Traitement d'une demande de souscription de bout en bout

1. **Déclenchement** : Un utilisateur soumet un formulaire de souscription. Le backend de l'application web lance une instance du processus "Souscription Client" via l'API de Camunda (**Vue des Processus**).
2. **Modélisation** : Le modèle BPMN du processus (**Vue Logique**) est instancié. La première tâche, "Vérifier l'éligibilité", est activée.
3. **Exécution** : L'Agent d'Éligibilité, développé en Python et s'exécutant dans un pod sur Kubernetes (**Vue de Développement** et **Vue de Déploiement**), récupère la tâche. Il exécute des règles métier et appelle une API externe.
4. **Décision et Flux** : L'agent complète la tâche avec un résultat "éligible". Le moteur d'orchestration suit le chemin nominal du processus vers la tâche "Créer le contrat".
5. **Compensation (si nécessaire)** : Si l'agent avait retourné "non éligible", le moteur aurait suivi un autre chemin, potentiellement vers une tâche "Envoyer notification de refus", illustrant la gestion des flux alternatifs. Si une étape ultérieure échouait (ex: paiement), le patron Saga serait invoqué pour annuler la création du contrat (**Vue des Processus**).
6. **Achèvement** : Le processus se poursuit à travers plusieurs agents (génération de contrat, paiement, notification) jusqu'à atteindre l'événement de fin. L'ensemble de la trace d'exécution est visible dans Operate (**Vue Logique** - Domaine de Gouvernance).

3.5.2. Scénario : Mise à l'échelle pour gérer un pic de demandes

1. **Situation** : Une campagne marketing génère un afflux massif de demandes de souscription. Le nombre d'instances de processus démarrées par minute augmente de manière significative.
2. **Observation** : Le nombre de tâches en attente pour le jobType "Vérifier l'éligibilité" augmente rapidement. Les métriques de monitoring (Prometheus) exposées par les agents et le moteur montrent une augmentation de la latence de traitement pour cette étape.
3. **Réaction Automatisée** : L'HorizontalPodAutoscaler (HPA) configuré pour le Deployment de l'Agent d'Éligibilité (**Vue de Déploiement**) détecte que l'utilisation moyenne du CPU des pods a dépassé le seuil cible (ex: 70%).
4. **Mise à l'échelle** : Kubernetes provisionne automatiquement de nouveaux pods pour l'Agent d'Éligibilité. Ces nouveaux pods démarrent, se connectent au moteur Camunda et commencent immédiatement à traiter les tâches en attente.
5. **Stabilisation** : Le nombre de workers ayant augmenté, le débit de traitement des tâches

augmente, la file d'attente se résorbe et la latence de traitement revient à la normale.
6. **Validation** : Ce scénario valide que l'architecture est élastique et répond à l'objectif de scalabilité. Le découplage entre le moteur et les agents permet de mettre à l'échelle uniquement le composant sous pression, optimisant ainsi l'utilisation des ressources.

La construction de ce DAT à travers le modèle 4+1 n'est pas un simple exercice de documentation. C'est un processus qui force l'alignement entre des équipes aux perspectives différentes. La Vue Logique et la Vue des Processus créent un consensus entre les analystes métier et les architectes. La Vue de Développement standardise les pratiques pour les équipes de développement. La Vue de Déploiement oblige les développeurs et les opérateurs à planifier concrètement l'exécution en production. Enfin, la Vue des Cas d'Usage agit comme un test d'intégration pour l'ensemble de la conception. Le document final est le produit de ce consensus, mais la valeur réside autant dans le processus collaboratif de sa création que dans l'artefact lui-même.

4. Perspectives Transversales

Cette section aborde les préoccupations qui ne sont pas confinées à un seul composant mais qui traversent l'ensemble de l'architecture. Ces aspects transversaux, tels que la sécurité, la gestion des données et la résilience, sont essentiels à la robustesse et à la viabilité à long terme de la plateforme.

4.1. Stratégie de Sécurité de Bout en Bout

La sécurité est intégrée à chaque couche de l'architecture, suivant une approche de "défense en profondeur". La stratégie repose sur le principe fondamental du Zero Trust.

4.1.1. Authentification et Autorisation (Zero Trust)

- **Principe** : Le modèle de sécurité Zero Trust est adopté, ce qui signifie qu'aucune confiance implicite n'est accordée à un acteur ou à un composant, qu'il soit interne ou externe au périmètre du réseau. Chaque demande d'accès doit être authentifiée et autorisée de manière explicite et continue.[73] Les trois piliers de cette approche sont :
  1. **Vérifier systématiquement** : Chaque tentative d'accès à une ressource est validée.

2. **Appliquer le moindre privilège** : Les accès accordés sont strictement limités au minimum nécessaire pour accomplir une tâche.[75]
3. **Supposer la compromission** : L'architecture est conçue en partant du principe que des attaquants sont déjà présents sur le réseau.[76]

- **Mise en œuvre** :
  - **Identité des Utilisateurs** : L'accès aux interfaces de la plateforme (Operate, Tasklist, Modeler) est protégé par une intégration avec le fournisseur d'identité (IdP) de l'entreprise via les protocoles OpenID Connect (OIDC) / OAuth 2.0. L'authentification multi-facteurs (MFA) est rendue obligatoire.
  - **Identité des Agents (Workloads)** : Chaque agent s'exécutant sur Kubernetes se voit attribuer une identité de charge de travail (ex: via SPIFFE/SPIRE ou un mécanisme équivalent du fournisseur cloud). Cette identité est utilisée pour s'authentifier auprès des autres composants de la plateforme (ex: Camunda, Confluent Cloud) et pour autoriser les interactions.
  - **Contrôle d'Accès Réseau** : Les Network Policies de Kubernetes sont utilisées pour segmenter le réseau au niveau micro. Par défaut, toute communication entre les pods est interdite. Seuls les flux explicitement autorisés (ex: un agent spécifique peut communiquer avec le Gateway Zeebe) sont permis, appliquant ainsi le principe de moindre privilège au niveau du réseau.[76]

4.1.2. Chiffrement des Données (en Transit et au Repos)

- **Principe** : Toutes les données sensibles, qu'elles soient en mouvement sur le réseau ou stockées sur un disque, doivent être chiffrées pour garantir leur confidentialité et leur intégrité.[77]
- **Mise en œuvre** :
  - **Données en Transit** :
    - **Trafic Externe** : Toute communication entrant ou sortant du cluster Kubernetes est chiffrée à l'aide du protocole TLS 1.2 ou supérieur. Cela inclut les appels API des clients, les interactions avec Confluent Cloud et les appels vers des services partenaires.
    - **Trafic Interne** : La communication entre les composants au sein du cluster Kubernetes sera également chiffrée. Dans une future itération (v2.0), un Service Mesh sera introduit pour appliquer le chiffrement TLS mutuel (mTLS) de manière transparente pour toutes les communications inter-services, garantissant l'authentification et le chiffrement de chaque appel.[79]
  - **Données au Repos** :
    - **Stockage Persistant** : Tous les volumes persistants utilisés par les composants (ex: logs de Zeebe) et les services de base de données managés (Elasticsearch, PostgreSQL) sont chiffrés au niveau du disque par le fournisseur de cloud, en

utilisant des algorithmes standards comme AES-256.[79]

- **Gestion des Clés** : Les clés de chiffrement pour les données au repos sont gérées via le service de gestion de clés (KMS) du fournisseur de cloud, avec une rotation régulière des clés. Pour les données les plus sensibles, une stratégie BYOK (Bring Your Own Key) peut être envisagée, où l'entreprise contrôle le matériel cryptographique.[80]

4.1.3. Gestion des Secrets

- **Principe** : Les informations d'identification sensibles (mots de passe, clés d'API, certificats TLS) ne doivent jamais être stockées en clair dans le code source, les fichiers de configuration ou les images de conteneurs. Elles doivent être gérées de manière sécurisée et injectées dans les applications au moment de l'exécution.
- **Mise en œuvre** :
  - **Stockage Centralisé** : Les secrets sont stockés dans un gestionnaire de secrets centralisé et sécurisé (ex: HashiCorp Vault, AWS Secrets Manager, Azure Key Vault). Ce service assure le chiffrement des secrets au repos, une gestion fine des accès (ACL) et une piste d'audit complète.
  - **Injection dans Kubernetes** : Les objets Secret natifs de Kubernetes sont utilisés comme un mécanisme d'abstraction, mais ils ne sont pas considérés comme une solution de stockage sécurisée (ils sont simplement encodés en base64).[81] Un opérateur Kubernetes, tel que l'External Secrets Operator, est déployé dans le cluster. Cet opérateur est responsable de :
    1. S'authentifier de manière sécurisée auprès du gestionnaire de secrets externe.
    2. Lire les secrets requis par les applications.
    3. Synchroniser ces secrets dans des objets Secret Kubernetes natifs, qui peuvent ensuite être montés en tant que volumes ou injectés en tant que variables d'environnement dans les pods des agents.[82]
  - **Rotation des Secrets** : Le gestionnaire de secrets centralisé est configuré pour effectuer une rotation automatique des informations d'identification à intervalles réguliers, réduisant ainsi la fenêtre d'opportunité en cas de compromission d'un secret.

4.2. Stratégie de Gestion des Données

Dans un système distribué et événementiel, la gestion des données présente des défis

uniques en matière de gouvernance, d'évolution et de cohérence.

4.2.1. Cycle de Vie des Données et Politiques de Rétention

- **Données de Processus** : L'historique d'exécution des processus stocké par Camunda 8 (dans Elasticsearch) est une donnée critique pour l'audit. Une politique de rétention est définie en fonction des exigences légales et métier. Par exemple, les données des processus terminés peuvent être conservées en ligne pendant 90 jours pour l'analyse opérationnelle, puis archivées dans un stockage à froid (ex: S3) pour une durée de 7 ans à des fins de conformité.
- **Événements** : Les événements stockés dans les topics Kafka de Confluent Cloud ont également une politique de rétention. Pour les topics représentant des flux de données en temps réel, une rétention courte (ex: 7 jours) peut suffire. Pour les topics servant de "source de vérité" (event sourcing), une rétention infinie (compacted topic) est configurée.

4.2.2. Gouvernance des Schémas et Évolution de l'Ontologie

- **Problème** : Dans une architecture événementielle, les événements sont des contrats de données publics. Une modification non coordonnée du schéma d'un événement par un producteur peut casser tous les consommateurs en aval.
- **Solution** : Le *Schema Registry* de Confluent Cloud est utilisé pour centraliser et gouverner les schémas de tous les événements circulant sur la plateforme.
    - **Format Standardisé** : Tous les événements doivent utiliser un format de sérialisation basé sur un schéma, tel qu'Apache Avro ou Protobuf.
    - **Validation des Schémas** : Des règles de compatibilité sont appliquées (ex: BACKWARD_COMPATIBILITY). Lorsqu'un producteur tente de publier un événement avec un nouveau schéma, le Schema Registry valide que ce nouveau schéma est compatible avec les versions précédentes. Si la modification est non compatible ("breaking change"), la publication est rejetée.
    - **Évolution Contrôlée** : Cette approche permet de faire évoluer l'ontologie des données de l'entreprise de manière contrôlée et sécurisée, en découplant le cycle de vie des producteurs de celui des consommateurs.

4.2.3. Modèle de Cohérence des Données (Eventual Consistency)

- **Principe** : L'architecture adopte un modèle de **cohérence à terme (eventual consistency)** pour la réplication des données entre les services. Cela signifie que le système garantit que si aucune nouvelle mise à jour n'est effectuée sur une donnée, toutes les lectures de cette donnée finiront par retourner la dernière valeur mise à jour. Cependant, une période de latence pendant laquelle les différentes répliques peuvent être temporairement incohérentes est acceptée.[84]
- **Justification** : Dans un système distribué à grande échelle, garantir une cohérence forte (où toutes les répliques sont toujours synchronisées) est extrêmement coûteux en termes de performance et de disponibilité (conformément au théorème CAP).[87] La cohérence à terme est un compromis pragmatique qui favorise la haute disponibilité et la tolérance aux partitions réseau, ce qui est essentiel pour la résilience de notre plateforme.[89]
- **Implications** : Les agents et les applications doivent être conçus en tenant compte de ce modèle. Par exemple, une interface utilisateur pourrait afficher des données potentiellement obsolètes pendant quelques secondes. Les opérations doivent être conçues pour être idempotentes, afin qu'une nouvelle tentative d'exécution n'entraîne pas de duplications de données. Il est important de noter que la cohérence à terme s'applique à la réplication des données ; la cohérence transactionnelle au niveau métier est, quant à elle, gérée par le patron Saga.

4.3. Stratégie de Résilience et de Continuité des Affaires

La résilience est la capacité du système à continuer de fonctionner en présence de défaillances. Elle est complétée par un plan de continuité des affaires qui définit comment récupérer après un sinistre majeur.

4.3.1. Analyse des Modes de Défaillance

Une analyse des modes de défaillance et de leurs effets (AMDE) est menée pour identifier les points de défaillance potentiels et leurs impacts :

- **Défaillance d'un pod d'agent** : Impact faible. Kubernetes redémarre automatiquement le pod. Les tâches en cours de traitement par ce pod verront leur verrou expirer et seront reprises par un autre pod.
- **Défaillance d'un nœud Kubernetes** : Impact moyen. Kubernetes replanifie les pods qui s'exécutaient sur le nœud défaillant sur d'autres nœuds sains. Une brève interruption de service pour les agents sur ce nœud est possible.

- **Défaillance d'une zone de disponibilité (AZ)** : Impact potentiellement élevé. C'est le scénario pour lequel la haute disponibilité est conçue.
- **Défaillance d'une région cloud entière** : Impact catastrophique. C'est le scénario pour lequel la reprise après sinistre est conçue.

4.3.2. Plan de Haute Disponibilité et de Basculement (Failover)

- **Objectif** : Assurer la continuité du service en cas de défaillance d'un ou plusieurs composants au sein d'une même région géographique. Le RTO (Recovery Time Objective) visé est proche de zéro.
- **Stratégie** :
  - **Déploiement Multi-AZ** : Tous les composants de l'infrastructure (cluster Kubernetes, bases de données managées) sont déployés de manière redondante sur au moins trois zones de disponibilité (AZ) au sein d'une même région cloud.
  - **Réplication des Données** : Les données critiques sont répliquées de manière synchrone ou quasi-synchrone entre les AZ. C'est le cas pour les logs de Zeebe, les données d'Elasticsearch et de PostgreSQL. Confluent Cloud gère également la réplication des données Kafka sur plusieurs AZ.
  - **Basculement Automatisé** : En cas de défaillance d'une AZ, les équilibreurs de charge redirigent automatiquement le trafic vers les AZ saines. Kubernetes replanifie automatiquement les pods sur les nœuds disponibles dans les autres AZ. Le basculement est conçu pour être entièrement automatisé et transparent pour les utilisateurs.[90]

4.3.3. Stratégie de Sauvegarde et de Reprise après Sinistre (Disaster Recovery)

- **Objectif** : Restaurer le service en cas de sinistre majeur affectant une région cloud entière (ex: catastrophe naturelle, panne généralisée). Le RTO visé est de quelques heures, et un certain niveau de perte de données (RPO - Recovery Point Objective) est acceptable.[91]
- **Stratégie** :
  - **Infrastructure-as-Code** : Le plan de reprise repose sur le principe de l'IaC. L'infrastructure complète peut être recréée à la demande dans une autre région à partir du code Terraform.
  - **Sauvegardes Inter-régions** : Des sauvegardes (snapshots) régulières de toutes les données persistantes (bases de données, volumes persistants) sont effectuées et répliquées de manière asynchrone vers une région cloud secondaire.

- **Procédure de Reprise** : En cas de sinistre, la procédure de reprise manuelle (ou semi-automatisée) consiste à :
  1. Déclarer le sinistre et activer le plan de reprise.
  2. Exécuter les scripts Terraform pour provisionner une nouvelle infrastructure dans la région secondaire.
  3. Restaurer les données à partir des dernières sauvegardes inter-régions.
  4. Mettre à jour les enregistrements DNS pour rediriger le trafic vers la nouvelle région.
  5. Valider le bon fonctionnement du système avant de le déclarer opérationnel.

5. Feuille de Route d'Évolution de l'Architecture

L'architecture présentée dans ce document (v1.0) est une fondation robuste conçue pour répondre aux besoins immédiats du projet pilote et des premières mises en production. Cependant, une architecture doit être un organisme vivant, capable d'évoluer pour répondre aux futurs défis techniques et métier. Cette section esquisse une feuille de route pour l'évolution de la plateforme.

5.1. État Actuel (v1.0 - Pilote)

La version 1.0 de l'architecture se concentre sur la validation des principes fondamentaux et la mise en place d'un socle stable et fonctionnel.

- **Périmètre** : L'architecture est entièrement basée sur l'**orchestration de processus**. Elle fournit les capacités essentielles de modélisation (BPMN), d'exécution (Camunda 8), de communication asynchrone (Confluent Cloud) et de déploiement conteneurisé (Kubernetes).
- **Forces** : La v1.0 excelle en termes de **gouvernance, de traçabilité et de contrôle centralisé**. Le choix de l'orchestration explicite rend les processus critiques transparents et faciles à gérer, ce qui est idéal pour les premières phases du projet où la maîtrise de la complexité est prioritaire.
- **Limitations** :
  - Le modèle est purement centré sur l'orchestration, ce qui peut être trop rigide pour certains cas d'usage très dynamiques ou décentralisés.
  - La sécurité et la gestion du trafic inter-services reposent sur les mécanismes de base de Kubernetes (Network Policies), qui peuvent s'avérer insuffisants à très grande échelle.

    - L'outillage pour la gestion opérationnelle des agents (visualisation de leur état, configuration dynamique) est rudimentaire et repose sur les outils standards de Kubernetes.

5.2. Évolution Future (v2.0 - Industrialisation)

La version 2.0 vise à industrialiser la plateforme en introduisant des capacités plus avancées pour répondre à une augmentation de l'échelle, de la complexité et de la diversité des cas d'usage.

5.2.1. Introduction de la Chorégraphie pour des Cas d'Usage Spécifiques

- **Objectif** : Introduire le patron de **chorégraphie** de manière complémentaire à l'orchestration, pour les cas d'usage où un couplage encore plus lâche et une plus grande autonomie des services sont souhaitables.
- **Justification** : L'orchestration est parfaite pour les processus transactionnels critiques. Cependant, pour des scénarios de type "notification" ou "fan-out", où un événement doit être diffusé à de multiples services qui réagissent de manière indépendante, la chorégraphie est plus simple et plus scalable.[33] Par exemple, lorsqu'une "Commande est expédiée", plusieurs services (facturation, analyse, communication client) peuvent vouloir réagir à cet événement sans que l'orchestrateur principal ait besoin de les connaître ou de les gérer explicitement.[33]
- **Mise en œuvre** : L'architecture v1.0 est déjà prête pour cela grâce à Confluent Cloud. Il s'agira de définir des topics Kafka publics pour les événements métier majeurs. Les processus orchestrés publieront ces événements à des moments clés, et des services "chorégraphiés" (qui peuvent être d'autres agents) s'y abonneront pour déclencher leurs propres logiques, créant ainsi une architecture hybride qui tire le meilleur des deux mondes.[56]

5.2.2. Intégration d'un Bus de Services (Service Mesh)

- **Objectif** : Déployer un service mesh (ex: Istio, Linkerd) au sein du cluster Kubernetes pour gérer de manière centralisée et transparente la communication inter-services.
- **Justification** : À mesure que le nombre d'agents et d'interactions augmente, la gestion

manuelle de la sécurité, de la fiabilité et de l'observabilité des communications devient complexe et sujette aux erreurs.[92] Un service mesh abstrait cette complexité de la logique applicative des agents.[94] Il apportera des bénéfices critiques :
    - **Sécurité** : Application automatique du chiffrement mTLS pour toutes les communications, sans modification du code des agents.[96]
    - **Fiabilité** : Gestion centralisée des politiques de nouvelles tentatives (retries), des délais (timeouts) et des disjoncteurs (circuit breakers).[97]
    - **Gestion du Trafic** : Capacités avancées de routage (ex: déploiements canary, A/B testing) pour des mises en production plus sûres.[94]
    - **Observabilité** : Collecte automatique de métriques, de journaux et de traces pour toutes les communications, fournissant une vue détaillée du graphe de dépendances entre les services.[95]
- **Mise en œuvre** : Le service mesh sera déployé sur le cluster Kubernetes. Chaque pod d'agent se verra injecter un conteneur "sidecar" (le proxy du service mesh) qui interceptera tout le trafic réseau entrant et sortant, appliquant les politiques définies de manière centralisée dans le "control plane" du mesh.[96]

5.2.3. Développement d'un Portail de Gestion des Agents

- **Objectif** : Créer une interface utilisateur centralisée (un portail) dédiée aux équipes d'exploitation et de développement pour gérer et superviser la flotte d'agents.
- **Justification** : Actuellement, la gestion des agents se fait via des outils de bas niveau comme kubectl ou les tableaux de bord de Kubernetes. Pour améliorer l'efficacité opérationnelle et permettre à des utilisateurs moins techniques d'interagir avec la plateforme, un portail dédié est nécessaire. Il s'inspirera des portails de gestion RH ou des portails de gestion publique qui centralisent l'information et les actions pour les agents.[98]
- **Fonctionnalités Vues** :
    - **Catalogue d'Agents** : Un registre de tous les agents disponibles, avec leur description, leur version, leur propriétaire (équipe) et la documentation de leurs capacités.
    - **Tableau de Bord Opérationnel** : Une vue en temps réel de l'état de chaque agent (nombre de répliques, consommation de ressources, taux d'erreur), avec des alertes.
    - **Gestion de la Configuration** : Une interface pour visualiser et, potentiellement, modifier certains paramètres de configuration des agents de manière sécurisée (ex: niveaux de log, seuils d'alerte).
    - **Actions Opérationnelles** : Des boutons pour déclencher des actions contrôlées, comme le redémarrage d'un groupe d'agents ou la mise à l'échelle manuelle en cas d'urgence.

6. Annexes

6.1. Glossaire Architectural

Terme	Définition
ADR (Architecture Decision Record)	Un document qui capture une décision architecturale importante, son contexte, et les conséquences de la décision.
Agent (Agentique)	Un système logiciel autonome qui perçoit son environnement, prend des décisions et agit pour atteindre des objectifs spécifiques.
BPMN (Business Process Model and Notation)	Une notation graphique standard pour la modélisation des processus métier, directement exécutable par un moteur de processus.
Chorégraphie	Un patron de coordination de services distribués où il n'y a pas de contrôleur central. Chaque service réagit aux événements publiés par d'autres.
Cohérence à Terme (Eventual Consistency)	Un modèle de cohérence pour les systèmes distribués qui garantit que, en l'absence de nouvelles mises à jour, toutes les répliques d'une donnée finiront par converger vers la même valeur.
External Task Worker	Un patron où des services externes ("workers") interrogent un moteur de

	processus pour récupérer et exécuter des tâches, permettant un découplage fort.
Infrastructure-as-Code (IaC)	La pratique de gérer et de provisionner l'infrastructure informatique à travers des fichiers de définition lisibles par machine, plutôt que par une configuration manuelle.
Kubernetes (K8s)	Une plateforme open-source d'orchestration de conteneurs pour automatiser le déploiement, la mise à l'échelle et la gestion des applications conteneurisées.
Orchestration	Un patron de coordination de services distribués où un contrôleur central (l'orchestrateur) dicte le flux des interactions et gère l'état du processus.
Saga	Un patron de gestion de la cohérence des données pour les transactions métier de longue durée qui s'étendent sur plusieurs microservices, utilisant des transactions de compensation pour annuler les actions en cas d'échec.
Service Mesh	Une couche d'infrastructure dédiée à la gestion de la communication entre les services, fournissant des fonctionnalités telles que la sécurité, la fiabilité et l'observabilité.
Zero Trust	Un modèle de sécurité qui part du principe qu'aucune confiance ne doit être accordée par défaut ; chaque demande d'accès doit être rigoureusement vérifiée.

6.2. Matrice de Conformité aux Exigences Non Fonctionnelles

Ce tableau final synthétise la manière dont l'architecture v1.0 répond à chaque objectif architectural clé défini dans la section 1.4, servant de liste de contrôle pour la validation de la conception.

Objectif Architectural	Exigence Clé	Solution Architecturale et Validation
Gouvernance et Auditabilité	Traçabilité complète de chaque instance de processus.	**Camunda 8 (Operate)** fournit un historique visuel et détaillé de chaque étape. **Journalisation centralisée** avec des ID de corrélation permet de lier les logs des agents au processus.
Évolutivité et Scalabilité	Gestion des pics de charge sans dégradation.	**Déploiement sur Kubernetes** avec **Horizontal Pod Autoscaler (HPA)** pour les agents. Le moteur **Zeebe** est scalable horizontalement. **Confluent Cloud** est un service managé élastique.
Résilience et Fiabilité	Tolérance aux pannes de composants.	**Déploiement Multi-AZ** pour tous les composants. **Patron External Task Worker** avec verrouillage et timeout pour la reprise des tâches. **Patron Saga** implémenté dans BPMN pour la cohérence métier.
Maintenabilité et Interopérabilité	Déploiement et évolution indépendants des agents.	**Patron External Task Worker** et **communication asynchrone** créent un découplage fort.

		Conteneurisation (Docker) permet le polyglottisme. **SDK partagé** standardise les interactions.

Ouvrages cités

1. L'IA agentique, qu'est-ce que c'est ? - Red Hat, dernier accès : octobre 1, 2025, https://www.redhat.com/fr/topics/ai/what-is-agentic-ai
2. Qu'est-ce que l'IA agentique ? | IBM, dernier accès : octobre 1, 2025, https://www.ibm.com/fr-fr/think/topics/agentic-ai
3. IA agentique et IA générative - Red Hat, dernier accès : octobre 1, 2025, https://www.redhat.com/fr/topics/ai/agentic-ai-vs-generative-ai
4. La stratégie de l'entreprise : emmener sa boîte vers le futur - CCI Business Builder, dernier accès : octobre 1, 2025, https://business-builder.cci.fr/guide-creation/le-business-model/agir-vision-mission-strategie
5. Recâbler l'entreprise avec l'IA agentique : réussir intelligemment | Concierto Insights & Agentic AI - Avrio, dernier accès : octobre 1, 2025, https://avriodata.ai/fr/resources/rewiring-the-enterprise-with-agentic-ai
6. L'IA agentique : l'utilité des agents d'IA dans le processus DevOps et la sécurité - GitLab, dernier accès : octobre 1, 2025, https://about.gitlab.com/fr-fr/topics/agentic-ai/
7. Les bases de la gouvernance de l'architecture d'entreprise - Conexiam, dernier accès : octobre 1, 2025, https://conexiam.com/fr/principes-de-base-de-la-gouvernance-de-larchitecture-dentreprise/
8. SOUTENABILITÉS ! - Haut-commissariat à la stratégie et au plan, dernier accès : octobre 1, 2025, https://www.strategie-plan.gouv.fr/files/2025-01/fs-2022-rapport-soutenabilites-mai_0_0.pdf
9. Document d'Architecture Technique - Awoui, dernier accès : octobre 1, 2025, https://www.awoui.com/post/document-d-architecture-technique
10. La nécessité et l'utilité d'écrire un Document d'Architecture Technique (DAT) dans une entreprise - Doknet, dernier accès : octobre 1, 2025, https://doknet.fr/referentiel-ais/administrer-et-securiser-les-infrastructures/appliquer-les-bonnes-pratiques-dans-ladministration-des-infrastructures/la-necessite-et-lutilite-decrire-un-document-darchitecture-technique-dat-dans-une-entreprise/
11. Tuckman's Stages of Group Development - WCU of PA - West Chester University, dernier accès : octobre 1, 2025, https://www.wcupa.edu/coral/tuckmanStagesGroupDelvelopment.aspx
12. Architecture cible - Secrétariat du Conseil du trésor - Gouvernement du Québec,

dernier accès : octobre 1, 2025, https://www.tresor.gouv.qc.ca/fileadmin/PDF/ressources_informationnelles/architecture_entreprise_gouvernementale/AEG3.0_Architecture_cible.pdf
13. 1. Introduction — Documentation VITAM - Architecture 3.14.2, dernier accès : octobre 1, 2025, https://www.programmevitam.fr/ressources/Doc3.14.2/html/archi/introduction.html
14. Guide pratique de l'architecture d'entreprise | Gouvernement du Québec, dernier accès : octobre 1, 2025, https://cdn-contenu.quebec.ca/cdn-contenu/adm/min/cybersecurite_numerique/Publications/AEG/guide-pratique-de-larchitecture-dentreprise-2.pdf
15. Architectural Blueprints—The “4+1” View Model of Software Architecture - UBC Computer Science, dernier accès : octobre 1, 2025, https://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf
16. Gouvernance de l'architecture d'entreprise | Le guide ultime - SAP LeanIX, dernier accès : octobre 1, 2025, https://www.leanix.net/fr/wiki/ea/gouvernance-architecture-enterprise
17. « Event Driven Architecture » : un modèle à fort enjeux. - Onepoint, dernier accès : octobre 1, 2025, https://www.groupeonepoint.com/fr/publications/event-driven-architecture-un-modele-darchitecture-repondant-aux-enjeux-daujourdhui-et-de-demain/
18. Benefits and Best Practices for Infrastructure as Code - DevOps.com, dernier accès : octobre 1, 2025, https://devops.com/benefits-and-best-practices-for-infrastructure-as-code/
19. Principes pour des infrastructures résilientes - UNDRR, dernier accès : octobre 1, 2025, https://www.undrr.org/media/78699/download
20. Découvrez tout le savoir-faire d'Orange Business | Orange Business, dernier accès : octobre 1, 2025, https://www.orange-business.com/fr
21. Architecture dès la conception - Dynamap, dernier accès : octobre 1, 2025, https://www.dynamap.fr/articles/architecture-des-la-conception/
22. L'IaC (Infrastructure-as-Code), qu'est-ce que c'est ? - Red Hat, dernier accès : octobre 1, 2025, https://www.redhat.com/fr/topics/automation/what-is-infrastructure-as-code-iac
23. Infrastructure as Code (IaC) : Principaux avantages et considérations pour les entreprises, dernier accès : octobre 1, 2025, https://www.okoone.com/fr/spark/technologies-et-innovation/infrastructure-as-code-iac-principaux-avantages-et-considerations-pour-les-entreprises/
24. Business Process Model and Notation - Wikipedia, dernier accès : octobre 1, 2025, https://en.wikipedia.org/wiki/Business_Process_Model_and_Notation
25. BPMN: Business Process Model and Notation - Introduction - Enterprise Architect, dernier accès : octobre 1, 2025, https://www.sparxsystems.eu/languages/bpmn/
26. BPMN Basics: Understanding the Building Blocks of Business Process Models, dernier accès : octobre 1, 2025, https://enterprisearchitecture.work/blog/2024/bpmn-basics-understanding-the-building-blocks-of-business-process-models/

27. BPMN 2.0 Symbols - A complete guide with examples. - Camunda, dernier accès : octobre 1, 2025, https://camunda.com/bpmn/reference/
28. Architecture pilotée par les événements : le guide définitif - AppMaster, dernier accès : octobre 1, 2025, https://appmaster.io/fr/blog/architecture-evenementielle
29. style d'architecture Event-Driven - Microsoft Learn, dernier accès : octobre 1, 2025, https://learn.microsoft.com/fr-fr/azure/architecture/guide/architecture-styles/event-driven
30. Camunda External Task Worker - The Basics - viadee Blog, dernier accès : octobre 1, 2025, https://blog.viadee.de/en/camunda-external-task-worker
31. External Tasks | docs.cibseven.org, dernier accès : octobre 1, 2025, https://docs.cibseven.org/manual/latest/user-guide/process-engine/external-tasks/
32. What is Infrastructure as Code? - IaC Explained - AWS, dernier accès : octobre 1, 2025, https://aws.amazon.com/what-is/iac/
33. Orchestration vs Choreography | Camunda, dernier accès : octobre 1, 2025, https://camunda.com/blog/2023/02/orchestration-vs-choreography/
34. Microservices Saga – Orchestration vs Choreography - SciForce, dernier accès : octobre 1, 2025, https://sciforce.solutions/blog/microservices-saga--orchestration-vs-choreography-p4w6adoed326u2oi80jajdkn
35. Distributed workflow in microservices (Orchestration vs Choreography) | by harish bhattbhatt, dernier accès : octobre 1, 2025, https://harish-bhattbhatt.medium.com/distributed-workflow-in-microservices-orchestration-vs-choreography-cf03cfef25db
36. Camunda External Task Pattern - Medium, dernier accès : octobre 1, 2025, https://medium.com/@dashedsouvik/camunda-external-task-pattern-fd84a29d9d3e
37. Writing good workers | Camunda 8 Docs, dernier accès : octobre 1, 2025, https://docs.camunda.io/docs/components/best-practices/development/writing-good-workers/
38. Job workers | Camunda 8 Docs, dernier accès : octobre 1, 2025, https://docs.camunda.io/docs/components/concepts/job-workers/
39. APIs et Architecture Événementielle : Le Duo Gagnant - Accetal, dernier accès : octobre 1, 2025, https://accetal.fr/apis-et-architecture-evenementielle-le-duo-gagnant/
40. Saga Design Pattern - Azure Architecture Center | Microsoft Learn, dernier accès : octobre 1, 2025, https://learn.microsoft.com/en-us/azure/architecture/patterns/saga
41. Understanding the Saga Pattern in Microservices Architecture | by Girish | Prepster - Medium, dernier accès : octobre 1, 2025, https://medium.com/prepster/understanding-the-saga-pattern-in-microservices-architecture-4ca4392d97d0
42. SAGA Design Pattern - GeeksforGeeks, dernier accès : octobre 1, 2025, https://www.geeksforgeeks.org/system-design/saga-design-pattern/

43. Pattern: Saga - Microservices.io, dernier accès : octobre 1, 2025, https://microservices.io/patterns/data/saga.html
44. Software Architecture: Orchestration vs Choreography | by Vitor Moschetta - Medium, dernier accès : octobre 1, 2025, https://medium.com/@vitormoschetta/software-architecture-orchestration-vs-choreography-25eb3d726906
45. Microservices : orchestration versus chorégraphie, dernier accès : octobre 1, 2025, https://h8l.io/blog/microservices-orchestration-versus-choregraphie
46. Agentic Orchestration & Automation Platform - Camunda, dernier accès : octobre 1, 2025, https://camunda.com/platform/
47. Using Camunda 8, dernier accès : octobre 1, 2025, https://docs.camunda.io/docs/components/
48. Get started with microservice orchestration - Camunda 8 Docs, dernier accès : octobre 1, 2025, https://docs.camunda.io/docs/guides/orchestrate-microservices/
49. About the Orchestration Cluster | Camunda 8 Docs, dernier accès : octobre 1, 2025, https://docs.camunda.io/docs/next/components/orchestration-cluster/
50. Event Streaming: How it Works, Benefits, and Use Cases - Confluent, dernier accès : octobre 1, 2025, https://www.confluent.io/learn/event-streaming/
51. Stream Processing: An Introduction - Confluent, dernier accès : octobre 1, 2025, https://www.confluent.io/learn/stream-processing/
52. Confluent | The Data Streaming Platform, dernier accès : octobre 1, 2025, https://www.confluent.io/
53. Streaming Data Pipelines - Confluent, dernier accès : octobre 1, 2025, https://www.confluent.io/streaming-data-pipelines/
54. Stream Governance on Confluent Cloud, dernier accès : octobre 1, 2025, https://docs.confluent.io/cloud/current/stream-governance/index.html
55. Vue d'ensemble | Kubernetes, dernier accès : octobre 1, 2025, https://kubernetes.io/fr/docs/concepts/overview/
56. Orchestration vs Choreography - Which is better? - Wallarm, dernier accès : octobre 1, 2025, https://www.wallarm.com/what/orchestration-vs-choreography
57. Déployer et gérer des applications avec Kubernetes | DevSecOps - Stephane Robert, dernier accès : octobre 1, 2025, https://blog.stephane-robert.info/docs/conteneurs/orchestrateurs/kubernetes/deployments/
58. Architecture et concepts de Kubernetes - ArcGIS Online, dernier accès : octobre 1, 2025, https://enterprise-k8s.arcgis.com/fr/11.2/deploy/kubernetes-concepts.htm
59. Kubernetes deployment overview - Camunda 8 Docs, dernier accès : octobre 1, 2025, https://docs.camunda.io/docs/self-managed/reference-architecture/kubernetes/
60. Camunda 8 reference architectures, dernier accès : octobre 1, 2025, https://docs.camunda.io/docs/self-managed/reference-architecture/
61. Streamlined Deployment with Camunda 8.8, dernier accès : octobre 1, 2025, https://camunda.com/blog/2025/03/streamlined-deployment-with-camunda-8-8/
62. 4+1 Architectural view model in Software | by Pasan Devin Jayawardene - Medium, dernier accès : octobre 1, 2025,

https://medium.com/javarevisited/4-1-architectural-view-model-in-software-ec407bf27258
63. 4+1 architectural view model - Wikipedia, dernier accès : octobre 1, 2025, https://en.wikipedia.org/wiki/4%2B1_architectural_view_model
64. www.techno-science.net, dernier accès : octobre 1, 2025, https://www.techno-science.net/glossaire-definition/Architecture-logicielle-page-3.html#:~:text=La%20vue%20logique%20permet%20d,(framework)%20%C3%A0%20sa%20disposition.
65. NFE107 - Cours U&ARSI 5 - Vision Informatique Logique - Architecture Logicielle, dernier accès : octobre 1, 2025, https://informatique.cnam.fr/fr/spip.php?pdoc4538
66. Définir l'architecture logicielle, dernier accès : octobre 1, 2025, http://tvaira.free.fr/dev/fiches/fiche-c1-modele_architecture.pdf
67. Un projet d'architecture de A à Z : Les différentes étapes - Kim Cloutier architecte, dernier accès : octobre 1, 2025, https://www.kcarchitecte.ca/espace-educatif/etapes-projet-architecture
68. Le plan d'exécution - tp.demain, dernier accès : octobre 1, 2025, https://tpdemain.com/module/le-plan-dexecution/
69. (Ré)organiser mon code - Cours, dernier accès : octobre 1, 2025, https://cours.brosseau.ovh/tp/organisation/introduction.html
70. Gestion du code source - Qu'est-ce que c'est et comment l'utiliser ? - Unity, dernier accès : octobre 1, 2025, https://unity.com/fr/topics/source-code-management
71. Veiller à la qualité de votre code et sa documentation - CNIL, dernier accès : octobre 1, 2025, https://www.cnil.fr/fr/veiller-la-qualite-de-votre-code-et-sa-documentation
72. 12 meilleurs outils de revue de code pour les développeurs (édition 2025) - Kinsta, dernier accès : octobre 1, 2025, https://kinsta.com/fr/blog/outils-revue-code/
73. Qu'et-ce que la sécurité zero-trust ? | Oracle Afrique, dernier accès : octobre 1, 2025, https://www.oracle.com/africa-fr/security/what-is-zero-trust/
74. Modèle de sécurité Zero Trust | Akamai, dernier accès : octobre 1, 2025, https://www.akamai.com/fr/glossary/what-is-zero-trust
75. Qu'est-ce que la sécurité Zero Trust ? | IBM, dernier accès : octobre 1, 2025, https://www.ibm.com/fr-fr/topics/zero-trust
76. Le modèle Zero Trust, qu'est-ce que c'est ? - Red Hat, dernier accès : octobre 1, 2025, https://www.redhat.com/fr/topics/security/what-is-zero-trust
77. Chiffrement des données en transit et au repos, dernier accès : octobre 1, 2025, https://reseau.uquebec.ca/system/files/documents/chiffrement-donnees-transit-repos-v1-20240711.pdf
78. Comment chiffrer des données ? | Oracle France, dernier accès : octobre 1, 2025, https://www.oracle.com/fr/security/comment-chiffrer-donnees/
79. Security - Chiffrement en transit - Google Cloud, dernier accès : octobre 1, 2025, https://cloud.google.com/docs/security/encryption-in-transit?hl=fr
80. Les pratiques de chiffrement dans l'informatique en nuage (cloud ..., dernier accès : octobre 1, 2025,

https://www.cnil.fr/fr/les-pratiques-de-chiffrement-dans-linformatique-en-nuage-cloud-public
81. Secrets | Kubernetes, dernier accès : octobre 1, 2025, https://kubernetes.io/fr/docs/concepts/configuration/secret/
82. Kubernetes : choisir la meilleure solution de gestion de secrets - Kaliop, dernier accès : octobre 1, 2025, https://www.kaliop.com/fr/gestion-secrets-kubernetes/
83. Kubernetes Secrets Management - Medium, dernier accès : octobre 1, 2025, https://medium.com/@MetricFire/kubernetes-secrets-management-70e0d269e813
84. What is eventual consistency? - Milvus, dernier accès : octobre 1, 2025, https://milvus.io/ai-quick-reference/what-is-eventual-consistency
85. Eventual consistency - Wikipedia, dernier accès : octobre 1, 2025, https://en.wikipedia.org/wiki/Eventual_consistency
86. Eventual Consistency in Distributed Systems | Learn System Design - GeeksforGeeks, dernier accès : octobre 1, 2025, https://www.geeksforgeeks.org/system-design/eventual-consistency-in-distributive-systems-learn-system-design/
87. What is eventual consistency? - Aerospike, dernier accès : octobre 1, 2025, https://aerospike.com/glossary/eventual-consistency/
88. How to understand eventual consistency in distributed systems? - Design Gurus, dernier accès : octobre 1, 2025, https://www.designgurus.io/answers/detail/how-to-understand-eventual-consistency-in-distributed-systems
89. What is Eventual Consistency? Definition & FAQs - ScyllaDB, dernier accès : octobre 1, 2025, https://www.scylladb.com/glossary/eventual-consistency/
90. Qu'est-ce que la continuité d'activité, la haute disponibilité et la ..., dernier accès : octobre 1, 2025, https://learn.microsoft.com/fr-fr/azure/reliability/concept-business-continuity-high-availability-disaster-recovery
91. Qu'est-ce que la reprise après sinistre ? | Google Cloud, dernier accès : octobre 1, 2025, https://cloud.google.com/learn/what-is-disaster-recovery?hl=fr
92. Service Mesh in Microservices - GeeksforGeeks, dernier accès : octobre 1, 2025, https://www.geeksforgeeks.org/system-design/service-mesh-in-microservices/
93. Service mesh 101: What is a service mesh? - Mulesoft, dernier accès : octobre 1, 2025, https://www.mulesoft.com/api/what-is-a-service-mesh
94. What is a service mesh? - Red Hat, dernier accès : octobre 1, 2025, https://www.redhat.com/en/topics/microservices/what-is-a-service-mesh
95. Service Mesh: The Integral Component of Microservices Architecture - [x]cube LABS, dernier accès : octobre 1, 2025, https://www.xcubelabs.com/blog/service-mesh-the-integral-component-of-microservices-architecture/
96. What is Service Mesh? - AWS, dernier accès : octobre 1, 2025, https://aws.amazon.com/what-is/service-mesh/
97. Service Mesh: Benefits, Challenges, and 7 Key Concepts - Tigera, dernier accès : octobre 1, 2025, https://www.tigera.io/learn/guides/service-mesh/

98. Portail RH : quels sont les avantages pour votre collectivité ? - Berger Levrault, dernier accès : octobre 1, 2025, https://www.berger-levrault.com/fr/parole-d-expert/pourquoi-mettre-en-place-un-portail-rh-dans-votre-collectivite/
99. Le portail de la fonction publique: Accueil, dernier accès : octobre 1, 2025, https://www.fonction-publique.gouv.fr/
100. Le portail de la gestion publique - collectivites-locales.gouv.fr, dernier accès : octobre 1, 2025, https://www.collectivites-locales.gouv.fr/finances-locales/le-portail-de-la-gestion-publique