Durable execution

Workflows qui survivent au crash, à l'attente, et à l'écart

AUTEUR Jérôme Vetillard

PUBLICATION 27 mai 2026

MAJ 29 mai 2026

SOURCE Twingital Institute

RAISE Article V · Architecture composite

PLANCHES PL.01 · PL.03

VALIDITY DOMAIN

Workflows agentiques en production qui nécessitent durabilité, replay, signals humains, ou compensation. Hors cadre : agents conversationnels courts sans état persistant, pour lesquels la durabilité serait disproportionnée.

SOMMAIRE

Durable execution

RESUME

La durable execution est ce qui distingue un agent qui survit aux crashes d'un agent qui les subit. Examine Temporal, DBOS, Restate, Inngest, Trigger.dev. Pose la séparation workflow et activity comme instanciation technique de la doctrine composite. Traite des patterns pratiques (agent-as-saga, human-in-the-loop comme event d'attente, gestion du payload LLM dans l'history). Conclut sur le choix DBOS pour les environnements régulés Postgres-natifs.

01 Le crash n’est pas une exception

Le crash d’un processus en production est, en environnement régulé, le test architectural le plus instructif. Il sépare en quelques secondes les systèmes qui survivent de ceux qui perdent leur audit, leur état, ou leur engagement vis-à-vis du signataire qui les avait initiés. Cette distinction n’est pas un détail d’ingénierie. Elle est la frontière qui sépare un agent industriellement déployable d’un agent expérimental qui ne franchira pas les exigences d’auditabilité de l’EU AI Act, du MDR, ou du PCCP de la FDA.

Dans une architecture classique sans durable execution, le crash d’un processus pendant un workflow long produit une rupture irrémédiable. Si l’agent était en train d’attendre une signature humaine au moment du crash, l’attente disparaît, le signataire ne reçoit jamais la requête, et le processus métier reste suspendu dans un état indéterminé. Si l’agent venait de débiter un compte et s’apprêtait à créditer le destinataire, le crédit ne se fait pas, l’audit montre un débit sans contrepartie, et l’incident est extrêmement coûteux à clore. Si l’agent venait de qualifier une prédiction toxicologique et s’apprêtait à l’inscrire dans la cohorte de qualification, l’inscription est perdue, le PSUR est faux, et la conformité MDR est compromise.

> La durable execution n’est pas une option, c’est l’infrastructure de la couche signée.

La durable execution est le paradigme technique qui rend ces ruptures impossibles. Elle garantit qu’un workflow en cours d’exécution est intégralement persisté, étape par étape, dans une infrastructure capable de le reprendre là où il en était, sur n’importe quelle machine, après n’importe quel crash. Le crash devient un événement banal, sans conséquence sur la continuité du processus métier. Cette propriété, qui semble triviale énoncée ainsi, exige une architecture spécifique que les frameworks d’orchestration classiques n’apportent pas.

Le présent cours expose la doctrine de la séparation workflow et activity, cartographie les cinq moteurs de durable execution disponibles en 2026, déploie les cinq patterns d’implémentation essentiels, traite spécifiquement de la gestion du payload LLM dans les workflows agentiques, et conclut sur la position du Twingital Institute en faveur de DBOS pour les environnements régulés Postgres-natifs.

02 La séparation workflow et activity

La doctrine architecturale centrale de la durable execution repose sur une distinction stricte entre deux types de code, séparés par un contrat formel et exécutés dans deux régimes distincts.

Le workflow est le code qui orchestre la séquence d'étapes d'un processus métier. Il est strictement déterministe. Son exécution peut être rejouée à l'identique depuis son historique. Il ne peut pas appeler directement d'API externe, d'horloge, de générateur aléatoire, ou de fonction non déterministe. Il ne peut que coordonner des appels à des activities et attendre leurs résultats. Sa durée d'exécution peut être de quelques secondes ou de plusieurs années. Sa résilience aux crashes est totale. Sa progression est journalisée dans une history persistante append-only.

L'activity est le code qui réalise les opérations effectives du processus. Elle peut appeler n'importe quelle API externe, lire et écrire dans les bases, invoquer un LLM, interroger un capteur. Elle est typiquement courte, idempotente (ou rendue idempotente par convention), et elle peut échouer. Quand elle échoue, le moteur de durable execution la retente selon une politique de retry configurée. Sa non-détermination n'est pas un problème, parce que son résultat est journalisé une fois obtenu et le workflow n'a pas besoin de la rejouer.

> Une activity peut échouer. Un workflow ne peut pas oublier.

Cette distinction n'est pas seulement une convention de programmation. Elle est portée par le moteur de durable execution lui-même, qui exécute le workflow dans un environnement contraint qui rejette toute opération non déterministe, et qui exécute les activities dans un environnement normal avec retries automatiques. Tenter de fusionner les deux mondes (appeler une API directement depuis un workflow, ou maintenir un état durable dans une activity) produit des comportements imprévisibles que le moteur signale comme non déterministe au moment du replay.

Le déterminisme du workflow est ce qui rend possible le mécanisme central de la durable execution : le replay. Quand le workflow reprend après un crash, le moteur recharge son history persisté et rejoue le code du workflow depuis le début. À chaque appel d'activity, le moteur vérifie si le résultat est déjà dans

l’history. S’il y est, il le retourne immédiatement sans réexécuter l’activity. S’il n’y est pas, il invoque l’activity. Cette mécanique garantit qu’aucune activity n’est exécutée deux fois lorsqu’elle a déjà réussi, et que le workflow continue exactement là où il en était au moment du crash.

> Ce qui n’est pas dans l’history du workflow n’a pas eu lieu pour le workflow.

Cette compression formule a une conséquence pratique majeure. Un workflow ne peut pas faire confiance à son monde extérieur sans inscrire le résultat dans son history. Si une activity retourne un résultat et que le workflow ne le persiste pas avant de poursuivre, alors un crash effacera ce résultat, et le replay rappellera l’activity. C’est généralement la sémantique souhaitée. Mais cela impose une discipline stricte : tout ce qui doit être conservé doit transiter par une activity et être inscrit dans l’history. Les variables locales du workflow sont gratuites parce qu’elles sont déterministes (elles sont recalculées au replay), mais les effets de bord doivent être encapsulés.

03 Cartographie des moteurs durables 2026

Le paysage des moteurs de durable execution s'est rapidement enrichi entre 2022 et 2026. Cinq solutions méritent un traitement détaillé, chacune avec ses forces, ses limites, et son contexte d'usage privilégié. Le choix du moteur est une décision architecturale structurante qui doit être faite à la conception, comme celle du broker événementiel exposée au cours 4, et qui est très coûteuse à révoquer en cours de route.

Temporal

Temporal est la référence historique de la durable execution moderne. Fondée en 2019 par Maxim Fateev et Samar Abbas (anciens créateurs du projet Cadence chez Uber, dont Temporal est un fork avec une licence permissive), la société Temporal Technologies a porté ce paradigme architectural dans l'industrie. Son produit éponyme est aujourd'hui déployé chez Snap, Coinbase, Datadog, Box, et de nombreuses entreprises pour qui la durabilité des processus est critique.

L'architecture Temporal repose sur trois composants. Les workers exécutent le code des workflows et des activities. Le frontend gère les requêtes des clients. Le persistence layer (Cassandra, PostgreSQL, MySQL) stocke l'history des workflows. Le moteur de matching coordonne workers et tâches. Cette architecture distribuée est mature, hautement scalable, et industriellement éprouvée.

Temporal supporte plusieurs langages clients (Go natif, Java, Python, TypeScript, .NET, PHP, Ruby), ce qui en fait le choix par défaut des programmes polyglottes. Sa licence MIT permet l'auto-hébergement sans contrainte. La version managée Temporal Cloud, opérée par Temporal Technologies, est disponible mais hébergée principalement aux Etats-Unis, ce qui pose les mêmes questions de souveraineté que Kafka Confluent Cloud abordées au cours 4.

Ses forces majeures sont la maturité (référence industrielle depuis sept ans en comptant Cadence), l'écosystème (clients multi-langages, SDK riches, observabilité native, replay tooling), et la communauté (large adoption, contribution active, ressources d'apprentissage abondantes). Sa faiblesse principale est la courbe d'apprentissage. Le modèle workflow et activity, le déterminisme strict, et le versioning des workflows constituent une discipline qui prend plusieurs semaines à intérioriser pour une équipe nouvelle.

Pour un système agentique en environnement régulé, Temporal est une valeur sûre à grande échelle. Son inconvénient spécifique en santé européenne est que sa couche de persistance Cassandra ou PostgreSQL devra être qualifiée HDS séparément, et que la complexité opérationnelle s'ajoute à la complexité métier du programme.

L'infrastructure Temporal souveraine en Europe est possible mais demande de l'effort. Le déploiement on-premises ou sur cloud souverain (OVH, Scaleway) est faisable, avec PostgreSQL comme persistance et Kubernetes pour l'orchestration des workers. La société française Spinaker propose en mai 2026 une offre Temporal managée en infrastructure souveraine, encore en phase de validation. Pour les programmes très sensibles, l'option Temporal Cloud reste à éviter parce qu'elle est opérée aux Etats-Unis.

Du point de vue de la mise en oeuvre concrète, Temporal expose plusieurs primitives utiles aux workflows agentiques. Les child workflows permettent à un workflow parent de lancer des workflows enfants qui s'exécutent indépendamment, utile pour décomposer une saga longue en sous-sagas autonomes. Les signals permettent à un workflow d'attendre l'arrivée d'un événement externe, ce qui réalise le pattern human-in-the-loop traité plus loin. Les queries permettent à un client externe d'interroger l'état courant d'un workflow sans le modifier. Les continue-as-new permettent à un workflow de redémarrer en réinitialisant son history, utile pour les workflows perpétuels qui accumuleraient sinon un history infini.

L'écosystème Temporal Cloud, Temporal Server, et Temporal SDK est cohérent et bien documenté. Pour une équipe qui démarre, le temps de prise en main est de l'ordre de trois à six semaines avant maîtrise opérationnelle. Cette maîtrise reste un investissement humain qu'il convient de planifier.

DBOS

DBOS, abréviation de DataBase Operating System, est le moteur de durable execution issu d'une recherche académique du MIT pilotée par Mike Stonebraker (lauréat du prix Turing 2014 pour ses travaux sur les bases de données relationnelles). La société DBOS Inc., spin-off du MIT en 2022, a commercialisé la doctrine sous-jacente : faire de PostgreSQL le système d'exécution durable lui-même, sans middleware d'orchestration distinct.

L'architecture DBOS repose sur une idée doctrinale forte. Toutes les structures de données dont un moteur d'orchestration distribué a besoin (history, queues, locks, semaphores) sont des tables PostgreSQL. L'exécution durable d'un work-

flow est une séquence de transactions PostgreSQL. La durabilité est portée par la durabilité de PostgreSQL elle-même, qui est éprouvée depuis trois décennies. L'exactly-once est garanti par les contraintes transactionnelles de PostgreSQL, sans avoir à inventer un nouveau protocole de consensus distribué.

Cette doctrine a quatre conséquences architecturales majeures, alignées avec les exigences de l'environnement régulé européen.

Première conséquence, la durabilité est aussi forte que celle de PostgreSQL. Pour un déploiement en santé européenne, cela signifie que la durabilité du workflow est portée par un PostgreSQL qui peut être qualifié HDS (offres OVH, Scaleway, Outscale). Le périmètre de qualification réglementaire de l'infrastructure de durabilité se réduit à la qualification de PostgreSQL, qui est déjà courante et bien documentée.

Deuxième conséquence, l'exactly-once est natif. Une activity DBOS s'exécute dans une transaction PostgreSQL. Si elle réussit, son effet est commit. Si elle échoue, son effet est rollback. Le moteur retente l'activity en sachant qu'elle n'a produit aucun effet partiel. Cette garantie est précieuse en santé clinique où une inscription au dossier patient ne doit jamais être doublée ni perdue.

Troisième conséquence, la simplicité opérationnelle. Pas de cluster distribué à opérer, pas de Cassandra à dimensionner. Un PostgreSQL standard, dimensionné pour la charge du programme, suffit. Cette simplicité réduit le coût d'exploitation et la surface d'attaque sécurité.

Quatrième conséquence, l'écosystème PostgreSQL est mobilisable. Tous les outils de monitoring, sauvegarde, réplication, et audit qui existent autour de PostgreSQL s'appliquent au moteur de durable execution. C'est un avantage opérationnel considérable pour les équipes qui ont déjà une expertise PostgreSQL.

Les faiblesses de DBOS sont sa jeunesse (le projet est en production depuis seulement deux ans) et son écosystème encore en construction. Les SDK officiels couvrent TypeScript et Python en mai 2026, avec Go et Java en cours de développement. Sa communauté est plus modeste que celle de Temporal. Pour les programmes très grande échelle qui dépassent les capacités d'un PostgreSQL single-node, DBOS peut nécessiter un déploiement Citus ou équivalent qui complique l'architecture.

> Postgres natif et exactly-once. Le reste suit.

Cette compression formule résume la position doctrinale du Twingital Institute en faveur de DBOS pour les programmes en environnement régulé européen. La section dédiée plus loin développe cette position.

Du point de vue de la mise en oeuvre, le modèle de programmation DBOS est volontairement minimaliste. Un workflow est une fonction TypeScript ou Python annotée. Les activities (appelées transactional functions en DBOS) sont des fonctions également annotées qui s'exécutent dans une transaction PostgreSQL. L'annotation indique au runtime DBOS d'inscrire l'invocation dans une table dédiée du même PostgreSQL, ce qui rend l'exécution durable de manière transparente pour le développeur.

L'exactly-once dans DBOS repose sur une mécanique élégante. Quand une transactional function est invoquée, son nom et ses paramètres sont hashés en une clé d'invocation unique. Le runtime DBOS commence par regarder dans la table d'historique si une invocation avec cette clé a déjà été commitée. Si oui, le résultat est retourné directement sans réexécution. Si non, la fonction s'exécute dans une transaction qui inclut l'inscription de son résultat dans la table d'historique. La transactionnalité PostgreSQL garantit l'atomicité : soit l'effet métier et l'inscription d'historique sont tous deux commitées, soit ni l'un ni l'autre. Cette mécanique transforme PostgreSQL en moteur de durable execution sans middleware additionnel.

L'inscription de l'historique dans le même PostgreSQL que les données métier a une conséquence opérationnelle puissante. Les requêtes d'audit peuvent joindre l'historique aux données métier dans une seule transaction SQL, ce qui permet de répondre à des questions complexes (par exemple, lister toutes les inscriptions au dossier patient X qui ont eu lieu via le workflow Y, avec leur état d'exécution complet) en une seule requête. Cette capacité est précieuse pour les audits réglementaires et les investigations d'incidents.

Restate

Restate, fondé en 2023 par d'anciens contributeurs d'Apache Flink dont Stephan Ewen, occupe une position originale dans le paysage. Sa proposition de valeur est la simplicité de l'expérience développeur (DX) sans renoncer à la rigueur de la durable execution.

L'architecture Restate repose sur un serveur Rust qui stocke les invocations et leurs progressions, et des SDK clients (TypeScript, Java, Go, Python, Kotlin) qui exposent une API simple. Le journaling des effets de chaque étape est automa-

tique et transparent pour le développeur. Restate introduit également les Virtual Objects, qui sont des entités stateful avec exécution séquentielle garantie, utiles pour modéliser des acteurs ou des entités à état longitudinal.

Restate a gagné de l'attention en 2024 et 2025 pour deux raisons. D'abord, son modèle de programmation est plus accessible que celui de Temporal, avec moins de cérémonies et un onboarding plus rapide. Ensuite, son architecture single-binary réduit la complexité opérationnelle par rapport à Temporal sans aller jusqu'à l'intégration Postgres de DBOS.

Ses faiblesses sont son adoption encore récente et sa maturité à confirmer pour les déploiements production critiques. La société est jeune, l'écosystème est en construction, et les cas d'usage industriels à grande échelle restent à publier. Pour les programmes nouveaux qui valorisent la productivité de l'équipe sur les premiers mois, Restate est une option intéressante. Pour les programmes critiques en environnement régulé où la maturité prime, l'attente d'une consolidation supplémentaire est prudente.

Inngest

Inngest, fondée en 2022 par Tony Holdstock-Brown, occupe le pôle serverless du paysage. Sa proposition de valeur est l'exécution durable de workflows sans gestion d'infrastructure, accessible via un SDK TypeScript principalement, avec un modèle de programmation orienté event-driven et step functions.

L'architecture Inngest est managée. Les fonctions de l'utilisateur s'exécutent sur ses propres serveurs (ou en serverless via Vercel, AWS Lambda, etc.), mais l'orchestration et la durabilité sont assurées par l'infrastructure Inngest. Cette séparation est élégante en termes de DX mais elle pose les questions de souveraineté habituelles. L'infrastructure Inngest est opérée aux Etats-Unis, ce qui exclut Inngest pour les déploiements santé européens régulés.

Inngest mérite considération pour les programmes non régulés qui valorisent la rapidité de mise en marché, l'absence de gestion d'infrastructure, et un SDK TypeScript idiomatique. Pour les programmes régulés, l'absence d'option auto-hébergeable est rédhibitoire.

Trigger.dev

Trigger.dev, fondée en 2023 par Eric Allam et Matt Aitken, est positionnée comme alternative open-source à Inngest avec une option d'auto-hébergement. Son SDK est TypeScript-natif, son modèle de programmation est centré sur les tasks, schedules, et events, et son DX vise l'élégance et la rapidité.

Trigger.dev gagne en adoption dans les écosystèmes JavaScript et TypeScript, particulièrement pour les workflows liés à des applications web modernes. Pour les programmes polyglottes, l'absence de SDK Python, Go, ou Java limite son adoption. Sa maturité reste à confirmer pour les programmes critiques.

Pour les programmes auxquels TypeScript suffit et qui valorisent la rapidité de développement avec option d'auto-hébergement, Trigger.dev est une option crédible. Pour les programmes critiques en santé régulée, l'attente d'une consolidation supplémentaire est prudente.

04 Cinq patterns essentiels

Au-delà du choix de moteur, la durable execution mobilise cinq patterns d'implémentation qui structurent la conception des workflows agentiques. Ces patterns sont en grande partie communs aux différents moteurs, avec des variations d'expression dans leurs SDK respectifs.

Agent-as-Saga

Le pattern Saga, exposé au cours 4 dans son acception générale, prend une forme spécifique en durable execution agentique. L'agent qui réalise un processus métier complexe est un workflow durable qui orchestre une séquence d'activities, dont certaines compensables. La compensation, en cas d'échec d'une étape, est elle-même une activity durable inscrite dans l'history. Cette inscription est essentielle : elle garantit que la compensation est attribuable, journalisée, et auditable, exactement comme l'opération qu'elle compense.

Pour ToxTwin V3.0, l'agent de qualification d'une nouvelle molécule est un workflow durable qui orchestre cinq étapes. Première étape, validation des données d'entrée et calcul des descripteurs moléculaires. Deuxième étape, réservation d'un slot GPU sur le pool de calcul. Troisième étape, exécution du modèle de prédiction de toxicité. Quatrième étape, comparaison du résultat à la cohorte de holdout figée (SHA256[0:16] = 47d3927ebc18cb7f) pour mesure de l'écart de promotion. Cinquième étape, inscription de la prédiction signée dans la cohorte de qualification courante. Si une étape échoue, les étapes précédentes sont compensées (libération du slot GPU, purge des descripteurs intermédiaires) et l'échec est inscrit dans l'audit avec sa raison typée.

Human-in-the-loop comme event d'attente

Le pattern central de l'agentique en environnement régulé est l'attente d'une signature humaine. L'agent ne signe pas, l'agent propose. Le signataire humain valide ou refuse, et c'est sa décision qui produit l'événement signé inscrit dans l'audit. Sans durable execution, ce pattern impose un compromis désagréable : soit l'agent fait du polling actif sur l'état de la signature (consommation continue de ressources, latence, complexité), soit l'agent maintient une connexion permanente (fragilité face aux crashes).

> Le workflow dort jusqu'à ce que l'humain signe.

Cette compression formule décrit le pattern propre à la durable execution. Le workflow durable, après avoir présenté une proposition au signataire, exécute un signal handler qui attend l'arrivée d'un événement de signature. Pendant cette attente, le workflow ne consomme aucune ressource active. Sa progression est gelée dans l'history. Si l'humain signe dans cinq minutes, le workflow reprend cinq minutes plus tard. Si l'humain signe dans trois jours, le workflow reprend trois jours plus tard, sur une machine potentiellement différente. Si l'humain ne signe jamais, un timeout configuré déclenche la compensation. Cette mécanique gérée par le moteur de durable execution est la réponse technique propre à la doctrine de l'unsigned agent exposée au cours 8.

Pour PREDICARE, le pattern human-in-the-loop prend la forme suivante. Un clinicien lance une consultation augmentée. Le workflow durable orchestre la préparation du contexte (lecture longitudinale, calcul des indicateurs, invocation du modèle d'aide à la décision). À la fin de cette préparation, le workflow présente au clinicien une proposition structurée (diagnostic suggéré, niveau de confiance, sources mobilisées, options de prise en charge). Le clinicien peut accepter la proposition telle quelle, la modifier, ou la refuser intégralement. Sa décision est inscrite via un signal qui contient le verdict typé, les éventuelles modifications, et la justification clinique. Le workflow reprend et inscrit la décision signée dans le dossier patient via une activity transactionnelle DBOS.

La durée d'attente du signal peut être très variable. Dans le cas usuel, le clinicien décide en quelques minutes au cours de la consultation. Dans des cas spécifiques (avis multidisciplinaire requis, complément d'examen demandé), la décision peut être différée de plusieurs jours. Le workflow durable supporte les deux régimes sans modification. Le timeout configuré (par exemple 7 jours) déclenche une compensation automatique qui clôt la consultation en l'absence de décision, libère les ressources mobilisées, et inscrit l'événement de timeout dans l'audit. Cette compensation est elle-même un acte qui doit être journalisé et qui peut faire l'objet d'analyse pour identifier les patterns de non-décision.

Long-running query et signal

Un workflow durable n'est pas une boîte noire. Il est interrogeable et adressable par deux mécanismes complémentaires. Les queries permettent à un client externe de lire l'état courant du workflow sans le modifier (par exemple, savoir à quelle étape il en est, quel est son écart de promotion intermédiaire, combien d'activities ont réussi). Les signals permettent à un client externe d'injecter un événement dans le workflow (par exemple, une signature humaine, une demande d'annulation, une modification de paramètres en cours de route).

Cette interface query plus signal transforme le workflow durable en machine à états distribuée et interrogeable. Pour un système agentique en environnement régulé, cette propriété est essentielle. Le tableau de bord de supervision interroge en temps réel l'état de chaque workflow en cours. Les signataires humains reçoivent des notifications et signalent leurs décisions. Les opérateurs de production peuvent intervenir sur des workflows en cours sans les redémarrer.

Versioning de workflow

Les workflows durables ont une particularité contraignante. Un workflow lancé il y a six mois et toujours en cours d'exécution doit pouvoir reprendre son cours, même si le code du workflow a été modifié depuis. Cette contrainte est gérée par le mécanisme de versioning des workflows, propre à chaque moteur de durable execution.

Temporal expose une primitive `workflow.getVersion()` qui permet au développeur d'introduire une logique conditionnelle selon la version du code en cours de replay. Les anciens workflows continuent d'exécuter l'ancienne logique, les nouveaux workflows exécutent la nouvelle. DBOS gère le versioning différemment, en stockant la version du code applicatif avec chaque workflow et en empêchant le replay de versions incompatibles, avec une stratégie de drain pour migrer progressivement.

Cette discipline de versioning est l'équivalent technique de la doctrine PCCP de la FDA exposée au cours 8. Les modifications d'un workflow déployé sont des modifications prévues qui doivent être documentées, qualifiées, et déployées avec un protocole explicite. Le moteur de durable execution rend cette discipline atteignable techniquement, mais elle reste une responsabilité de l'équipe applicative.

Gestion du payload LLM dans l'history

Le pattern le plus spécifique à la durable execution agentique concerne la taille des payloads. Un workflow durable inscrit dans son history le résultat de chaque activity. Une activity qui appelle un LLM produit typiquement un résultat de plusieurs kilooctets (réponse texte complète, métadonnées, traces de raisonnement). Une seule conversation longue avec un LLM peut produire plusieurs centaines de kilooctets cumulés dans l'history.

Pour Temporal, dont l'history est stocké entièrement dans la base de persistance, cette accumulation devient problématique. Au-delà de quelques mégaoctets d'history, les performances de replay se dégradent, et le coût de stockage explose. Trois techniques permettent de contrôler la taille de l'history.

Première technique, le pointer pattern. L'activity stocke le payload volumineux dans un object store externe (S3, MinIO sur infrastructure souveraine) et ne retourne au workflow que l'identifiant du payload stocké. L'history contient seulement l'identifiant, qui est petit. Si le workflow a besoin du payload, il appelle une activity de lecture qui résout le pointer.

Deuxième technique, la summarisation. L'activity qui appelle le LLM retourne au workflow non pas la réponse complète, mais une summarisation typée et compacte (verdict, code de classification, score de confiance, identifiants extraits). La réponse complète est stockée ailleurs si elle doit être conservée (typiquement dans l'audit trail événementiel exposé au cours 4). Cette technique impose une discipline d'extraction structurée mais elle est souvent souhaitable indépendamment de la taille de l'history.

Troisième technique, le chunking de workflow. Au lieu d'un seul workflow durable qui maintient toute la conversation LLM, plusieurs workflows courts chaînés transmettent leur résultat via le broker événementiel. Chaque workflow a un history modeste. La conversation globale est reconstituée par projection à partir des événements.

Pour DBOS, dont la durabilité est portée par PostgreSQL, ces problématiques se posent différemment. PostgreSQL gère bien les payloads volumineux dans des colonnes JSONB ou TEXT, mais les mêmes principes d'hygiène architecturale s'appliquent. Le pointer pattern reste utile pour les payloads dépassant quelques mégaoctets.

Trois autres considérations méritent attention sur la gestion du payload LLM.

Première considération, la séparation entre le payload de raisonnement (chain-of-thought, traces, scratchpad) et le payload de décision (verdict structuré). Le raisonnement est volumineux mais rarement nécessaire pour le replay du workflow. La décision est compacte mais essentielle. Architecturalement, on inscrit la décision dans l'history du workflow (pour la replayabilité) et le raisonnement dans le journal d'audit événementiel (pour l'auditabilité réglementaire). Cette séparation respecte l'EU AI Act article 14 sur la supervision humaine, qui exige que la justification d'une décision soit accessible à l'auditeur, mais qui n'exige pas qu'elle soit dans le workflow lui-même.

Deuxième considération, le caching des invocations LLM. Une activity qui appelle un LLM avec exactement les mêmes inputs (prompt, contexte, paramètres de température) devrait retourner le même résultat lors d'un replay. Mais le LLM est non déterministe et peut produire des sorties différentes selon les versions du modèle et les seeds aléatoires. Le moteur de durable execution résout ce problème en cachant le résultat de l'activity dans l'history dès sa première exécution. Le replay relit le résultat caché sans réinvoquer le LLM. Cette propriété est essentielle pour la stabilité des audits, parce qu'un audit qui rejouerait le workflow et obtiendrait des résultats différents serait irrecevable.

Troisième considération, la rotation des modèles. Quand un modèle LLM est mis à jour par son fournisseur (cas typique des modèles hosted comme Claude, GPT, Mistral), les nouveaux workflows utiliseront le nouveau modèle, mais les workflows en cours doivent continuer avec le modèle utilisé lors de leur démarrage. Cette discipline impose de versionner explicitement le modèle dans l'invocation de l'activity et de pinning le modèle sur la durée de vie du workflow. Le moteur de durable execution facilite cette discipline par sa garantie de replay, mais elle reste une responsabilité applicative.

05 Le choix DBOS pour les environnements régulés Postgres-natifs

La position doctrinale du Twingital Institute sur le choix du moteur de durable execution en santé régulée européenne est tranchée. Pour les programmes nouveaux qui démarrent en 2026 sans contrainte d'écosystème préexistant, DBOS est le choix recommandé. Cette position mérite argumentation, parce qu'elle rompt avec le consensus industriel qui place Temporal en référence par défaut.

Premier argument, la qualification réglementaire de l'infrastructure de durabilité. Temporal nécessite la qualification HDS de sa couche de persistance (PostgreSQL, MySQL, ou Cassandra) plus la qualification de l'environnement d'exécution des workers, plus la qualification du composant Temporal Server lui-même. Cette triple qualification est faisable mais coûteuse en temps et en documentation. DBOS nécessite la qualification HDS de PostgreSQL uniquement, qui est déjà disponible via les offres souveraines françaises (OVH HDS, Scaleway HDS, Outscale). Le périmètre de qualification est divisé par trois, et le coût d'instruction réglementaire avec lui.

Deuxième argument, l'exactly-once natif. En santé clinique, une inscription au dossier patient ne doit jamais être ni doublée ni perdue. DBOS garantit cette propriété par la transactionnalité PostgreSQL, sans qu'aucun code applicatif n'ait à implémenter de logique d'idempotence. Temporal supporte également l'exactly-once mais au prix d'une discipline plus stricte côté application (rendre chaque activity idempotente via des clés d'idempotence, des contraintes uniques côté DB, des locks distribués). Cette discipline est facilement violée par inadvertance, et le débogage des doublons en production est coûteux.

Troisième argument, la simplicité opérationnelle. DBOS s'opère comme un PostgreSQL standard. Les équipes santé qui ont déjà PostgreSQL sous gestion HDS (la majorité) n'ajoutent pas de nouveau composant à leur stack. Temporal ajoute un cluster distribué dont l'opération nécessite des compétences spécifiques (configuration de Cassandra ou MySQL en haute disponibilité, dimensionnement des workers, observabilité Temporal). Pour un programme santé de taille moyenne (mille à dix mille workflows en parallèle), Temporal est sur-dimensionné et DBOS est exactement calibré.

Quatrième argument, l'écosystème PostgreSQL mobilisable. Les outils d'audit, de sauvegarde, de réplication, de monitoring PostgreSQL (pg_audit, wal-g, repmgr, pg_stat_statements) s'appliquent directement à DBOS. Cette mobilisation immédiate de l'écosystème PostgreSQL réduit le coût de mise en production et améliore la qualité opérationnelle.

Les contre-arguments à DBOS doivent être considérés honnêtement. Sa maturité est moindre que celle de Temporal (deux ans en production contre sept). Ses SDK couvrent moins de langages (TypeScript et Python uniquement en mai 2026). Sa communauté est plus modeste. Sa scalabilité au-delà du PostgreSQL single-node nécessite Citus ou une architecture similaire qui complexifie le déploiement. Pour les programmes très grande échelle (cent mille workflows simultanés ou plus), Temporal reste préférable. Pour les programmes polyglottes (Go, Java, .NET, PHP), Temporal couvre plus de langages.

La position du Twingital Institute n'est donc pas qu'un programme régulé doit toujours choisir DBOS. Elle est qu'un programme régulé européen Postgres-natif de taille moyenne (mille à cinquante mille workflows simultanés), en TypeScript ou Python, et démarrant en 2026, devrait choisir DBOS par défaut, et n'opter pour Temporal qu'en présence d'un argument explicite contre (très grande échelle, multi-langages obligatoire, écosystème Temporal préexistant). L'inversion de la charge de la preuve est doctrinale : en santé régulée européenne, l'argument à porter est celui de Temporal, pas celui de DBOS.

Cette inversion doctrinale est cohérente avec une orientation plus large du Twingital Institute. La sophistication architecturale n'est pas une valeur en soi. La signabilité, la conformité, et l'auditabilité sont des valeurs. Quand une solution plus simple atteint ces propriétés, elle est préférable à une solution plus complexe qui les atteint au prix d'une charge cognitive et opérationnelle plus élevée. DBOS atteint la durabilité avec PostgreSQL seul, là où Temporal demande un cluster spécialisé. La supériorité technique de Temporal en débit maximal (centaines de milliers de workflows par seconde sur un cluster bien dimensionné) est réelle mais non pertinente pour la majorité des programmes santé régulés, qui opèrent à des échelles plus modestes (dizaines à centaines de workflows par seconde).

Un cas typique illustre. Un programme de qualification de SaMD en France, taille moyenne, traite environ trois mille consultations augmentées par jour. Chaque consultation est un workflow durable de quelques minutes en moyenne, avec attente humaine pouvant durer jusqu'à quelques heures. Le pic à mille workflows simultanés est atteint en heure de pointe matinale. Cette charge tient confortа-

blement sur un PostgreSQL HDS de taille modeste (16 vCPU, 64 GB RAM, stockage NVMe). Temporal sur la même charge nécessiterait un cluster de trois nœuds Cassandra, plus trois nœuds Temporal Server, plus PostgreSQL pour les workflows métadonnées, plus les workers applicatifs. Le ratio coût d'infrastructure et complexité d'exploitation est de l'ordre de un à cinq en faveur de DBOS. À performance et propriétés de signabilité strictement équivalentes pour le périmètre considéré.

06 Articulation à l’architecture composite signable

La durable execution est l’instanciation technique de la couche déterministe de l’architecture composite signable exposée au cours 8. Le workflow durable est le composant qui porte la signabilité parce qu’il est strictement déterministe, rejouable, et inscrit dans une history append-only. L’activity est le composant qui encapsule l’appel à la couche non déterministe (LLM, modèles d’inférence, services externes). Le port de promotion est typiquement une étape spécifique du workflow, qui transforme une proposition issue d’une activity en un acte signé via l’invocation d’un signataire humain par signal.

Cette articulation a deux conséquences architecturales fortes.

Première conséquence, le workflow durable et le journal d’audit événementiel exposé au cours 4 sont complémentaires, pas redondants. Le workflow durable porte l’état d’exécution d’une instance de processus (où en est le workflow X au moment T). Le journal d’audit événementiel porte les faits signés inscrits dans l’histoire du système (qui a signé quoi à quel moment). Les deux sont alimentés par les mêmes activities, via le pattern Outbox qui garantit la cohérence transactionnelle entre l’écriture dans l’history du workflow et la publication de l’événement signé dans le broker.

Deuxième conséquence, la mesure de l’écart de promotion au sens de l’Article VI RAISE est nativement portée par les workflows durables. L’activity qui invoque le LLM produit une proposition. L’activity qui invoque le signataire humain produit la décision finale (promotion, écart, refus). La différence entre les deux est journalisée dans l’history du workflow. Une projection régulière agrège ces différences pour produire la distribution des écarts de promotion attendue par la doctrine.

Pour PREDICARE, le workflow durable orchestre une consultation augmentée multi-acteurs. Un patient arrive en consultation. Le clinicien lance une session via une commande qui crée un workflow durable. Le workflow appelle des activities pour récupérer le dossier longitudinal du patient, calculer des indicateurs de risque, invoquer un modèle d’aide à la décision, présenter une proposition au clinicien, attendre sa décision via signal, et inscrire la décision signée dans le dossier patient via une activity transactionnelle. Si le clinicien interrompt la consultation au milieu (urgence, appel téléphonique, fin de plage horaire), le workflow attend. Le clinicien peut reprendre la consultation plus tard, le workflow reprend exactement où il en était.

Pour ToxTwin V3.0, le workflow durable de qualification d'une prédiction toxicologique est plus court mais tout aussi structurant. Le workflow démarre quand une nouvelle molécule entre dans le pipeline de qualification. Une première activity calcule les descripteurs moléculaires (graphes, propriétés physico-chimiques, fingerprints). Une deuxième activity invoque le modèle GNN (graph neural network) v3.0 sur les descripteurs, et obtient une prédiction de toxicité avec son intervalle de confiance. Une troisième activity confronte cette prédiction à la cohorte de holdout figée (SHA256[0:16] = 47d3927ebc18cb7f) pour calculer l'écart de promotion attendu. Une quatrième activity présente le résultat structuré à un responsable scientifique habilité, qui valide ou refuse via signal. Une cinquième activity inscrit la prédiction signée dans la cohorte de qualification courante, dans une transaction PostgreSQL qui garantit l'exactly-once.

La durée totale du workflow est généralement de quelques minutes en mode automatisé (validations préalablement déléguées au pipeline de signature), de quelques heures à quelques jours quand un examen humain est requis. Le workflow durable supporte les deux régimes. La mesure de l'écart de promotion sur l'ensemble des prédictions qualifiées sur une période donnée est dérivée par projection à partir de l'history accumulée. Cette projection alimente le tableau de bord PCCP du dispositif, conformément à la doctrine FDA exposée au cours 8.

Une propriété spécifique de ce workflow mérite attention. Le moteur GNN v3.0 est versionné dans le workflow. Toute prédiction inscrite porte la version du modèle qui l'a produite. Si le modèle est mis à jour à v3.1, les nouveaux workflows utilisent v3.1, mais les workflows déjà inscrits restent attachés à v3.0. Cette discipline rend l'audit rétroactif possible. Un auditeur peut, à n'importe quel moment, rejouer le workflow d'une prédiction donnée avec exactement le modèle qui l'a produite, et vérifier que le résultat est reproductible.

07 Articulation à l'EDA

Le workflow durable et l'EDA sont deux couches complémentaires de l'architecture composite signable. La séparation de leurs rôles est nette.

Le workflow durable porte l'orchestration d'une instance individuelle de processus métier. Il a son propre identifiant, son propre history, sa propre durée de vie. Il est le sujet d'audit à l'échelle de l'instance.

L'EDA porte le transport et l'inscription des événements signés produits par les workflows. Le broker événementiel est l'archive doctrinale du système. Le journal d'audit est la projection événementielle qui sert à l'auditeur.

La cohérence transactionnelle entre l'history du workflow et la publication événementielle est portée par le pattern Outbox, exposé au cours 4. Quand une activity du workflow doit à la fois inscrire un fait dans le workflow et publier l'événement correspondant dans le broker, elle utilise une transaction PostgreSQL (en DBOS) ou un mécanisme équivalent (en Temporal avec une activity Outbox dédiée) qui garantit que les deux écritures sont commitées ensemble ou rollback ensemble. Cette discipline évite les divergences entre l'état d'exécution et l'audit événementiel.

L'articulation pratique en DBOS est élégante. L'activity transactionnelle qui produit un fait métier inscrit simultanément dans la même transaction PostgreSQL le fait métier (par exemple, l'inscription au dossier patient), l'inscription dans l'history DBOS du workflow (gérée automatiquement par le runtime), et un événement dans une table outbox dédiée. Un poller distinct (souvent un autre workflow durable DBOS) lit la table outbox et publie effectivement les événements dans Kafka ou NATS. Si la publication réussit, l'événement est marqué comme publié dans l'outbox. Si la publication échoue temporairement, l'événement reste dans l'outbox et sera publié au prochain passage du poller. Cette mécanique garantit que tout fait inscrit dans le workflow finit dans le broker, et inversement, qu'aucun événement publié dans le broker n'existe sans correspondance dans le workflow.

En Temporal, l'articulation est plus indirecte parce que l'history Temporal est stocké séparément des données métier. Le pattern recommandé combine une activity Outbox spécifique qui écrit dans une table outbox PostgreSQL applicative, et une autre activity qui poll cette table pour publier dans le broker. La cohérence transactionnelle est portée par la transaction applicative entre l'écri-

ture métier et l'écriture outbox, pas par l'history Temporal lui-même. Cette différence architecturale est l'un des arguments en faveur de DBOS pour les programmes où la cohérence cross-systèmes est critique.

La saga distribuée exposée au cours 4 prend en durable execution sa forme la plus rigoureuse. La saga orchestrée est un workflow durable qui pilote des activities transactionnelles avec compensations explicites. La saga chorégraphiée est une coordination entre plusieurs workflows durables qui réagissent à des événements du broker et émettent leurs propres événements. Dans les deux cas, le moteur de durable execution garantit que chaque étape (incluant les compensations) est exécutée au moins une fois et que son résultat est journalisé.

Pour les sagas chorégraphiées spécifiquement, l'articulation broker plus workflow durable mérite un développement. Un workflow durable peut s'abonner à un topic du broker via une activity spécialisée qui exécute en boucle un consommateur. Quand un événement arrive, l'activity retourne au workflow qui décide de l'action suivante. Cette mécanique permet à un workflow durable d'attendre indéfiniment des événements externes sans consommer de ressources actives, exactement comme il attend les signaux humains. Pour les architectures où plusieurs workflows durables coordonnent leur action via le broker, cette propriété est centrale.

08 Limites assumées

Quatre limites de la doctrine de durable execution exposée ici doivent être assumées explicitement.

Premièrement, la durable execution introduit un coût opérationnel et une complexité conceptuelle qui ne sont pas justifiés pour les workflows triviaux. Un agent qui répond à une requête synchrone unique sans état persistant n'a pas besoin de durable execution. Forcer un moteur de durable execution dans ce cas ajoute de la latence, de l'infrastructure, et de la complexité sans bénéfice. La règle pratique du Twingital Institute : la durable execution s'impose dès qu'un workflow comporte une attente humaine, une étape réversible, plus de trois activities, ou une durée d'exécution dépassant quelques secondes en moyenne.

Deuxièmement, la courbe d'apprentissage du modèle workflow et activity est réelle. Les équipes habituées au modèle synchrone classique doivent apprendre à raisonner sur le déterminisme, le replay, la non-détermination encapsulée dans les activities, le versioning des workflows. Cet apprentissage prend plusieurs semaines de pratique encadrée. Sa sous-estimation est la première cause d'échec des migrations vers la durable execution. Le Twingital Institute recommande un pilote initial sur un cas d'usage simple avant la généralisation.

Troisièmement, le paysage des moteurs évolue. Les versions citées dans le présent cours (Temporal, DBOS 2024-2026, Restate 1.x, Inngest 2024+, Trigger.dev 3.x) sont datées de mai 2026. Les fonctionnalités, les arbitrages comparatifs, et les positions doctrinales peuvent évoluer dans les 18 prochains mois. La position en faveur de DBOS pour les environnements régulés Postgres-natifs sera réévaluée tous les deux trimestres en cohérence avec la matrice de décision frameworks du cours 2.

Quatrièmement, la durable execution ne dispense pas de la gouvernance des contrats événementiels et de la discipline RAISE. Un workflow durable bien conçu sur DBOS ou Temporal peut être déployé dans une architecture composite signable rigoureuse, ou dans une architecture désordonnée. Le moteur de durable execution apporte la primitive technique de la couche déterministe, il ne garantit pas à lui seul la signabilité. La signabilité reste une propriété émergente de l'architecture composite, comme exposé au cours 8.

Le présent cours expose la couche workflow comme primitive technique. Le cours 6 traite des protocoles d'interopérabilité (MCP, A2A, ACP, UCP) qui constituent la couche transport au-dessus des workflows. Le cours 7 traite de l'observabilité eval-driven qui mesure le comportement des workflows en production.

Ensemble, ces trois cours (4, 5, et 6) constituent le triptyque technique au-dessus duquel l’architecture composite signable exposée au cours 8 prend sa forme opérationnelle.

A propos de cette serie

Ce cours appartient a la serie Architectures agentiques 2026 publiee par le Twingital Institute. La serie aborde l'architecture des systemes agentiques en environnement regule a partir du Framework RAISE.

L'ensemble de la serie, les planches d'architecture interactives, et la matrice de decision frameworks sont accessibles sur le site de l'Institut : twingital-ventures.com/fr/cours/ https://twingital-ventures.com/fr/cours/

TITRE	Durable execution
SOUS-TITRE	Workflows qui survivent au crash, à l'attente, et à l'écart
AUTEUR	Jérôme Vetillard
SOURCE	Twingital Institute
SERIE	Architectures agentiques 2026 · Cours 5 / 8
PUBLICATION	27 mai 2026
MAJ	29 mai 2026
RAISE	Article V · Architecture composite
PLANCHES	PL.01 · PL.03
MOTS-CLES	durable-execution · Temporal · DBOS · Restate · workflow · saga · human-in-the-loop
URL	https://twingital-ventures.com/fr/cours/2026-05-cours-5-durable-execution-fr/