+33 (0)1 48 10 09 46
formation@ambient-it.net
140 Avenue Jean Lolive 93500 Pantin - France

Mis à jour le 30/07/2026

S'inscrire

Formation Spring Batch

2 jours (14 heures)

Présentation

Spring Batch est le framework officiel de l'écosystème Spring, conçu pour exécuter et automatiser les traitements par lots (batchs) massifs, performants et fiables au sein des applications d'entreprise.

L'un des piliers de Spring Batch est sa gestion native de la robustesse et de l'état : il fournit un modèle structuré orienté chunk (lecture, traitement, écriture) associé à un système de métadonnées persistant. Grâce à cette architecture, le framework garantit la reprise automatique sur incident, la gestion stricte des transactions et la traçabilité complète des exécutions sans rejouer les étapes déjà validées.

Notre formation Spring Batch couvre à la fois les concepts fondamentaux du traitement par lots (Jobs, Steps, Tasklets, JobParameters) et les fonctionnalités avancées du framework telles que la tolérance aux fautes (skip et retry), le passage dynamique de paramètres (@StepScope), l'aiguillage de flux et les stratégies d'optimisation par le parallélisme (multi-threading et partitionnement).

À l'issue de cette formation, vous saurez concevoir, sécuriser et tester des batchs complexes prêts pour la production. Vous saurez isoler vos composants de test, gérer les fichiers corrompus ou erreurs réseau sans interrompre vos traitements, et orchestrer des flux volumineux optimisés pour de fortes contraintes de performance.

Comme toutes nos formations, celle-ci vous présentera **la dernière version stable** de la technologie et ses nouveautés.

Objectifs

- Situer Spring Batch dans un système d'information et maîtriser son vocabulaire (Job, Step, JobInstance, JobExecution, StepExecution, JobParameters, ExecutionContext).

- Mettre en place un projet Spring Boot avec Spring Batch et lancer un job via JobOperator.
- Écrire un TaskletStep pour une action unitaire, et un Step orienté chunk (ItemReader / ItemProcessor / ItemWriter) pour traiter du volume.
- Lire et écrire les formats courants : fichier plat (CSV), base de données (JPA), JSON.
- Exploiter les métadonnées persistées par Spring Batch (tables BATCH_*) pour superviser les exécutions et comprendre les mécanismes de relance (restart).
- Paramétrer un job et propager ces paramètres jusqu'aux composants du chunk (late binding, @StepScope), et valider les paramètres avant démarrage.
- Rendre un traitement tolérant aux fautes : skip, retry, seuils de rejet, traçabilité des lignes rejetées via un SkipListener.
- Piloter le déroulement d'un job : distinguer BatchStatus et ExitStatus, construire un flow conditionnel (.on/.to/.from, terminaisons, jokers, JobExecutionDecider).
- Accélérer un traitement en choisissant la bonne forme de parallélisme : split() de flows, step multi-thread (taskExecutor), partitionnement — et en identifier les pièges (état partagé, thread-safety, reprise).
- Tester un batch de façon fiable : composants en isolation, step isolé, job complet.
- Concevoir de bout en bout un batch de production complet (projet fil rouge final).

Public visé

- Développeurs Java/Kotlin, développeurs back-end, architectes logiciels
- lead techniques amenés à écrire ou reprendre des traitements de masse (imports/exports de fichiers, alimentation de bases, reprises de données, traitements de nuit)

Pré-requis

- Connaissance pratique du framework Spring / Spring Boot (injection de dépendances, starters, configuration par @Bean, application.properties).
- Maîtrise du langage Java ou Kotlin.
- Notions de SQL et de persistance (JPA/Hibernate) pour les travaux pratiques d'alimentation de base.
- Aucune connaissance préalable de Spring Batch n'est requise.

Pré-requis logiciels

- IntelliJ IDEA installé sur le poste de travail

Programme de notre formation Spring Batch

[Jour 1 - Matin]

Qu'est-ce qu'une application batch ?

- Les cas d'usage du traitement par lots : imports/exports massifs, alimentation de référentiels, reprises de données, traitements de nuit, relances clients.
- Ce qu'apporte un framework de batch par rapport à un simple script : reprise sur incident, transactions, supervision, traçabilité.
- Positionnement de Spring Batch dans l'écosystème Spring.

Architecture et vocabulaire

- Job et Step : le plan de traitement réutilisable et ses étapes.
- Les deux natures de Step : TaskletStep (action unitaire) et Step orienté chunk (lecture / traitement / écriture par lots).
- Le modèle d'exécution : JobInstance (exécution logique identifiée par le nom du job et ses paramètres d'identification), JobExecution (une tentative), StepExecution.
- JobParameters : leur rôle dans l'identité d'une instance.
- ExecutionContext : la carte persistée qui permet la reprise.
- Les composants d'infrastructure : JobRepository, JobOperator.

Mise en place du projet

- Les dépendances et les starters nécessaires.
- Configuration : base de métadonnées, initialisation du schéma, désactivation du lancement automatique au démarrage.

Le TaskletStep

- L'interface Tasklet et sa méthode execute().
- Le contrat RepeatStatus : FINISHED / CONTINUABLE, et la transaction par appel.
- Les cas d'usage : suppression de fichier, appel de webservice, procédure stockée.

[Jour 1 - Après-midi]

Assembler un Step et un Job, puis le lancer

- Construction d'un Step avec StepBuilder (nom, JobRepository, gestionnaire de transactions).
- Construction d'un Job avec JobBuilder et enchaînement des étapes.
- Lancement avec JobOperator ; récupération des jobs par injection de dépendances.
- TP 1 — HelloWorld
  - Mise en place du projet, écriture d'un premier Tasklet, assemblage du Step et du Job, lancement et lecture du résultat.

Les paramètres d'un job

- Créer et typer des JobParameters au lancement.

- Paramètres identifiants et non identifiants : conséquence directe sur l'identité de la JobInstance et donc sur la relance.
- Première façon de lire un paramètre : depuis le contexte d'exécution d'un Tasklet.
- TP 2 — Paramètres
  - Faire passer un paramètre au job, l'exploiter dans le Tasklet et provoquer volontairement un échec pour observer un statut FAILED.

Les tables de métadonnées (BATCH_*)

- Le modèle persisté
- Comment l'identité d'un job est calculée (nom + paramètres, résumés dans JOB_KEY).
- Pourquoi deux lancements avec les mêmes paramètres visent la même instance, et le rôle d'un paramètre technique (timestamp) pour rendre chaque appel unique.
- Lecture des compteurs de traitement portés par chaque StepExecution.

Enchaîner plusieurs steps et gérer la relance

- Enchaînement séquentiel et propagation de l'échec : les steps suivants ne sont pas lancés.
- Le comportement au redémarrage : un step COMPLETED est sauté.
- allowStartIfComplete : forcer le rejeu d'un step (nettoyage, préparation).
- startLimit : borner le nombre de tentatives.
- TP 3 — Multiple Step
  - Créer un job de plusieurs steps, provoquer un échec, relancer et constater la reprise à l'étape fautive ; forcer le rejeu d'un step déjà terminé.

Le traitement par chunk

- Le principe : N items lus et traités un par un, puis écrits en une transaction.
- Le chunk comme commit-interval, et son impact sur la performance et la mémoire.
- Ce que Spring Batch prend en charge seul : transactions courtes, streaming, sauvegarde de la position pour la reprise, compteurs read / write / commit.
- ItemReader : fichier plat (CSV), base en JDBC, base en JPA, JSON/XML, reader personnalisé, reader composite.
- ItemProcessor : transformation, filtrage (retour null), validation, processor composite.
- ItemWriter : fichier plat, base en JDBC, base en JPA, JSON/XML, writer composite.
- Assemblage d'un Step orienté chunk avec typage des items d'entrée et de sortie.
- TP 4 — Chunk
  - Importer un fichier CSV de ventes en base : écrire le DTO de lecture, le mapping vers l'entité et le calcul métier, observer l'effet de la taille de chunk sur les commits.
- TP 5 — Sens inverse
  - Exporter la table vers un fichier CSV trié, avec ligne d'en-tête et ligne de total (lecture paginée en base, agrégation, écriture fichier).

[Jour 2 - Matin]

Passer des paramètres dans toutes les couches

- Le late binding : lire un JobParameter par expression SpEL dans un Reader, un Processor ou un Writer.
- @StepScope et @JobScope : pourquoi ils sont indispensables, et ce qui casse sans eux.
- Le piège des types de retour : déclarer une interface plutôt qu'une classe concrète pour que le step reconnaisse le composant comme un flux à ouvrir et fermer.
- Valider les paramètres avant démarrage : JobParametersValidator et DefaultJobParametersValidator (paramètres requis / optionnels), et leurs limites.
- Les listeners : JobExecutionListener (beforeJob / afterJob) et StepExecutionListener (beforeStep / afterStep).
- TP 6 — Chunk paramétré
  - Piloter un même job par quatre paramètres lus chacun à un endroit différent de la chaîne : le fichier source (reader), un seuil de filtrage et une règle de calcul (processor), le format de sortie CSV ou JSON (writer).
  - Mise en place d'un validateur de paramètres.

Tolérance aux fautes : skip et retry

- Le comportement par défaut : une exception fait échouer le step, et les conséquences sur un traitement de masse.
- Activer le mode tolérant, et l'ordre d'intervention retry puis skip.
  - retry : rejouer une erreur passagère (indisponibilité, timeout) et borner les tentatives.
  - Skip : ignorer une ligne fautive et poursuivre ; le seuil de rejet (skipLimit) comme garde-fou qualité — au-delà, le fichier est mauvais et le job doit échouer.
- Déclarer finement les exceptions concernées (skip / noSkip).
- Le cas particulier du writer, qui reçoit un lot et bascule ligne par ligne.
- SkipListener : tracer les lignes rejetées dans un fichier de rejets exploitable par la production.
- Lecture des compteurs après incident (read, write, commit, rollback, skip).
- TP 7 — Fichier corrompu
  - Rendre un import tolérant à trois familles d'erreurs (ligne illisible, règle métier violée, panne passagère), fixer un seuil de rejet, produire un fichier de rejets, puis comparer un fichier acceptable et un fichier à refuser.

BatchStatus et ExitStatus

- BatchStatus : le statut interne qui pilote la reprise et les transitions (COMPLETED, FAILED, COMPLETED WITH SKIPS...).
- ExitStatus : le code de sortie personnalisable, destiné au reporting et à l'aiguillage.
- Produire un ExitStatus métier depuis afterStep (interface ou annotation).
- La propagation de l'ExitStatus du dernier step au job.
- TP 8 — ExitStatus
  - Produire trois codes de sortie métier différents à partir d'un même step selon un indicateur, et vérifier que le step reste en succès tout en portant un statut nuancé.

Conditionner le déroulement du job (flow)

- Les limites de l'enchaînement linéaire.
- Les transitions .on(...).to(...) et .from(...) : aiguiller selon l'ExitStatus.
- Les terminaisons : .end() (fin anticipée en succès), .fail() (échec explicite), .stopAndRestart(step) (mise en pause et point de reprise).

- Absorber un échec géré : router sur .on("FAILED") et terminer le job en succès.
- Les jokers ? et * et la règle du motif le plus spécifique.
- Les flows imbriqués pour composer des enchaînements complexes.
- JobExecutionDecider : aiguiller sur une décision calculée plutôt que sur un ExitStatus.

[Jour 2 - Après-midi]

Parallélisme (1/3)

- Exécuter plusieurs flows indépendants en parallèle et la jonction avant la suite.
- Le rôle du TaskExecutor qui fournit les threads.
- Composer plusieurs split() successifs.
- TP 9 — Flow
  - Construire une dizaine de jobs d'aiguillage de complexité croissante : séquentiel, deux branches, fin anticipée, échec explicite, réaction à l'échec, jokers, pause et reprise, flow imbriqué, reproduction d'un enchaînement à partir d'un schéma puis à partir d'un cahier des charges rédigé, decider, et parallélisation par split().

Parallélisme (2/3) : le step multi-thread

- Paralléliser le traitement des items d'un même step avec un TaskExecutor.
- Ce qui reste mono-thread et ce qui devient concurrent : conséquences concrètes sur le reader, le processor et le writer.
- Le vrai point de vigilance : l'état partagé d'un processor, et sa correction par des structures thread-safe.
- Dimensionner le pool d'exécution : taille, file d'attente, politique de saturation.
- TP 10 — Chunk parallélisé
  - Paralléliser un import dont le traitement est lent, mesurer le gain, constater le bug de concurrence introduit par un compteur partagé, puis le corriger.

Parallélisme (3/3) : le partitionnement

- Le principe maître / esclave : découper les données en partitions traitées en parallèle, chacune avec son propre reader.
- Écrire un Partitioner et alimenter le contexte d'exécution de chaque partition.
- Le step esclave et le late binding de sa tranche de données.
- Configurer le step maître : partitionneur, taille de grille, exécuteur.
- L'atout majeur : la reprise partition par partition après incident.
- Choisir entre split(), step multi-thread et partitionnement.
- TP 11 — Partitionnement
  - Importer dix fichiers en parallèle (une partition par fichier), vérifier l'isolation des partitions et le gain de temps, provoquer l'échec d'une seule partition et constater qu'elle seule est rejouée au redémarrage.

Tester un batch

- Tester un composant en isolation, sans contexte Spring (reader, processor).
- Tester un composant qui a besoin du contexte Spring (writer).
- Tester un composant à portée de step qui lit un paramètre de job : créer un contexte de StepExecution pour valider la résolution du late binding.
- Isoler l'exécution d'un seul step, avec une vraie StepExecution et un vrai comportement transactionnel.
- Tester un job complet et asserter sur les statuts, les compteurs et le chemin réellement parcouru plutôt que sur la console.
- Neutraliser les dépendances externes d'un batch dans les tests.
- TP 12 — Tester un batch (exercice inversé)
  - Le job est fourni et fonctionnel : c'est la classe de test qui est à écrire. Valider un reader en isolation (en-tête correcte, en-tête dans le désordre, nombre de colonnes invalide) puis le job complet de fusion de fichiers sans doublon.

Projet de synthèse

- Analyse d'un besoin métier complet et découpage en steps.
- Choix argumentés : Tasklet ou chunk, seuil de tolérance, forme de parallélisme, isolation des appels externes.
- TP PROJET FINAL — Batch de planification de camions
  - Réalisation d'un batch de production complet : contrôle de présence et de conformité du fichier de commandes, import tolérant aux fautes avec seuil de refus et fichier de rejets, enrichissement par un service externe (majoration des quantités selon la météo de chaque ville), génération des fichiers de chargement pour l'entrepôt et pour chaque chauffeur, puis archivage et purge. Le TP mobilise l'ensemble des notions du cours.

Pour aller plus loin

Sociétés concernées

Cette formation s’adresse à la fois aux particuliers ainsi qu’aux entreprises, petites ou grandes, souhaitant former ses équipes à une nouvelle technologie informatique avancée ou bien à acquérir des connaissances métiers spécifiques ou des méthodes modernes.

Positionnement à l'entrée en formation

Le positionnement à l'entrée en formation respecte les critères qualité Qualiopi. Dès son inscription définitive, l'apprenant reçoit un questionnaire d’auto-évaluation nous permettant d’apprécier son niveau estimé sur différents types de technologies, ses attentes et objectifs personnels quant à la formation à venir, dans les limites imposées par le format sélectionné. Ce questionnaire nous permet également d’anticiper certaines difficultés de connexion ou de sécurité interne en entreprise (intraentreprise ou classe virtuelle) qui pourraient être problématiques pour le suivi et le bon déroulement de la session de formation.

Méthodes pédagogiques

Stage Pratique : 60% Pratique, 40% Théorie. Support de la formation distribué au format numérique à tous les participants.

Organisation

Le cours alterne les apports théoriques du formateur soutenus par des exemples et des séances de réflexions, et de travail en groupe.

Validation

À la fin de la session, un questionnaire à choix multiples permet de vérifier l’acquisition correcte des compétences.

Sanction

Une attestation sera remise à chaque stagiaire qui aura suivi la totalité de la formation.